User's Manual

Appendices

SIMPROCESS

Release 5

CACI

EVER VIGILANT

Copyright © 2002-2015 CACI, INC.-FEDERAL.

All rights reserved. No part of this publication may be reproduced by any means without written permission from CACI.

The information in document is believed to be accurate in all respects. However, CACI cannot assume the responsibility for any consequences resulting from the use thereof. The information contained herein is subject to change. Revisions to this publication or new editions of it may be issued to incorporate such change.

SIMPROCESS is a registered trademark of CACI, INC.-FEDERAL.

Organization of the SIMPROCESS Documentation Set 6

Appendix A
Importing Version 2.2.1 Models 9

Import Procedures 10
Preparing Your Model For Import 11
Graphical Import Results 14
Properties Import Results 21
Importing Document Files 24

Appendix B
Activity Summary Table 25

Appendix C
SIMPROCESS File Structure 28

Appendix D
Statistical Distributions 35

Uniform Distribution 37
Normal Distribution 38
Triangular Distribution 39
PertBeta Distribution 40
Exponential Distribution 41
Gamma Distribution 42
Beta Distribution 43
Erlang Distribution 44
Weibull Distribution 45
Lognormal Distribution 46
Poisson Distribution 47
Hyper Exponential Distribution 48
Uniform Integer Distribution 49

Geometric Distribution **50**
Pareto Distribution **51**
Binomial Distribution **52**
Negative Binomial Distribution **53**
Inverse Gaussian Distribution **54**
Inverted Weibull **55**
Johnson SB Distribution **56**
Johnson SU Distribution **57**
Log-Logistic Distribution **58**
Log-Laplace Distribution **59**
Pearson Type V Distribution **60**
Pearson Type VI Distribution **61**
Random Walk Distribution **62**
Empirical Distribution **63**

Appendix E
Statistical Tools Glossary ***64***

Appendix F
SIMPROCESS System Attributes and Methods ***70***

System Attributes **71**
SIMPROCESS System Methods **81**
System Method Examples **126**
SIMPROCESS Color Table **168**

Appendix G
External Event Files ***169***

General Rules for Event Files **170**
Event Record Description **172**
Examples **174**

Appendix H
Simulation Results File ***175***

Format of the Simulation Results File **176**

Appendix I
UML Interfaces ***184***

Exporting to UML **186**
Rose Use Cases **188**

Appendix J
Running Models Without GUI ***191***

Appendix K
SIMPROCESS and External Java Classes ***194***

Java Extension Mechanism **195**
Model-Specific Java Additions **196**
ext Directory **197**

Organization of the SIMPROCESS Documentation Set

The SIMPROCESS documentation set consists of four manuals:

- *Getting Started With SIMPROCESS*
- *SIMPROCESS User's Manual*
- *SIMPROCESS Metadata Manual*
- *SIMPROCESS OrgModel Manual*

All of the manuals can be opened directly from the **Help/SIMPROCESS Manuals** menu. Also, each manual includes this section which provides links to the chapters in every manual. Press the Control key when clicking any link to open the linked file in a new window. For Windows systems, in order for links between manuals to work properly, Adobe Acrobat or Acrobat Reader must be used to view the manuals.

Getting Started

The *Getting Started With SIMPROCESS* manual is a must for first time SIMPROCESS users. This manual can also be used for evaluation purposes. The chapters are

- Business Process Modeling With SIMPROCESS
- Installation

- Building Your First Model With SIMPROCESS
- Evaluating Alternatives With SIMPROCESS
- Demonstration and Reference Models

User's Manual

The *User's Manual* is divided into four parts with each part being a separate file. Part A is an excellent reference for beginners and casual users. This part contains detailed documentation of the basic and intermediate functions of SIMPROCESS. The chapters are

- Process Modeling and Analysis with SIMPROCESS
- SIMPROCESS Basics
- Statistical Modeling Constructs
- Activity Modeling Constructs
- Resource Modeling Constructs
- Graphical Modeling Constructs
- Activity-Based Costing
- Statistical Output Reports

Part B is a reference intended for advanced users of SIMPROCESS. This part contains detailed documentation of the programming and library management functions in SIMPROCESS Professional Edition. The chapters are

- Reusable Templates and Libraries
- Customizing a Model with Attributes and Expressions
- More Advanced Model Building
- Exporting Results

Part C describes the integrated tools included with SIMPROCESS Professional. The chapters are

- Advanced Data Analysis
- SIMPROCESS Database
- Experiment Manager
- OptQuest for SIMPROCESS
- SIMPROCESS Dashboards
- Model Bundles
- Custom Reports
- Scenarios

The Appendices are

- Importing Version 2.2.1 Models
- Activity Summary Table
- SIMPROCESS File Structure
- Statistical Distributions
- Statistical Tools Glossary
- SIMPROCESS System Attributes and Methods
- External Event Files
- Simulation Results File
- UML Interfaces
- Running Models Without GUI
- SIMPROCESS and External Java Classes

Metadata Manual

The Metadata Manual describes how to build and edit SIMPROCESS metamodels, assign metamodels to a SIMPROCESS model, and enter metadata in a SIMPROCESS model. The chapters are

- SIMPROCESS Metadata
- SIMPROCESS Metamodel Editor
- Assigning Metamodels
- Entering Metadata
- BPEL Metadata

OrgModel Manual

The OrgModel Manual describes how to build and edit SIMPROCESS Organization and Resource Models (OrgModels) and assign OrgModels to a SIMPROCESS model. The chapters are

- SIMPROCESS Organization and Resource Models
- SIMPROCESS OrgModel Editor
- Assigning OrgModels
- Using OrgModels with SIMPROCESS

APPENDIX A

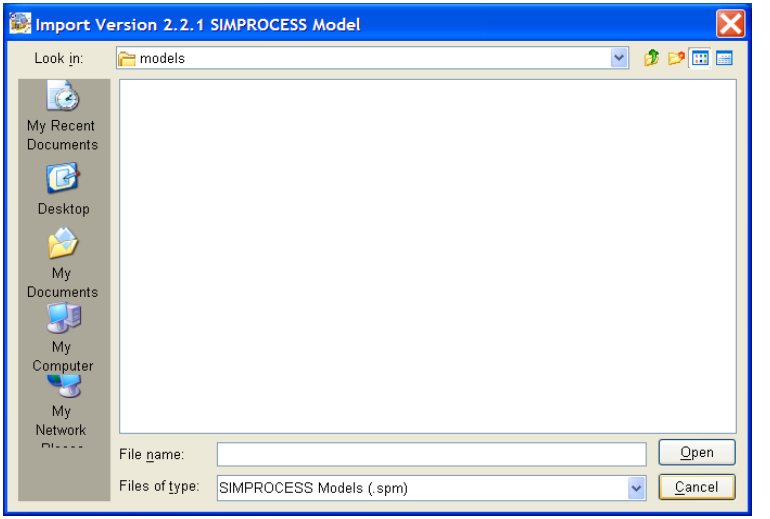
Importing Version 2.2.1 Models

Version 2.2.1 or 2.2.2 models cannot be opened using **File/Open**. Earlier version models must be imported to version 4. This is done using **File/Import/Version 2.2.1 Model...** Model versions earlier than 2.2.1 cannot be imported. These models must be saved in 2.2.1 or 2.2.2 before they can be imported into version 4. However, it is recommended that models be saved in 2.2.3 before importing. SIMPROCESS 2.2.3 is a special Student version (Import Utility) that is designed to prepare models for import to version 4. The 2.2.3 Student version can load and save models from versions of SIMPROCESS earlier than 2.2.1. The special Student version of SIMPROCESS 2.2.3 is available at www.simprocess.com.

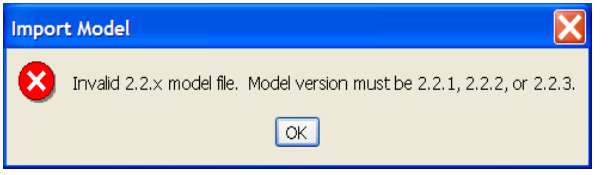
Earlier version models must be imported because the structure of SIMPROCESS and the structure of a SIMPROCESS model file has changed. Version 3 and higher model files are XML files. Thus, they can be read by any XML parser.

Import Procedures

To import a model select **File/Import/Version 2.2.1 Model...** and choose the model to import.



If the file selected is a version 3 or higher file, it will open. However, if the file is pre-2.2.1, an error dialog will appear.



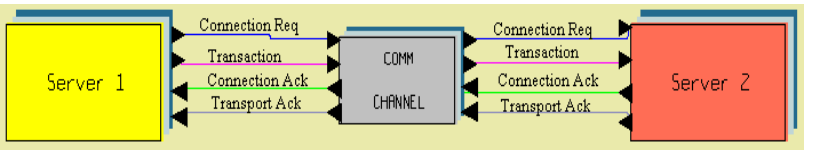
Preparing Your Model For Import

The import procedure reads a 2.2.1, 2.2.2, or 2.2.3 model file and converts the information to XML that can be read by SIMPROCESS version 4 or higher. However, due to limitations in 2.2.1 and 2.2.2 files, some preparation may be required. For larger models, it is recommended that they be opened and saved in 2.2.3. Without being saved in 2.2.3, importing the properties of an Activity, Connector, Pad, etc. from earlier version models depends upon the name of that item. Therefore, if there are duplicate names at the same level, errors will occur, and the conversion will not be complete. Depending on the version of your model, there are three or four primary steps in preparing your model for import. The first item only applies to models that have not been saved in 2.2.3.

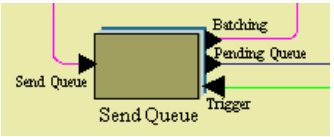
- Make sure there are no duplicate names at the same level (2.2.1 and 2.2.2 only).
- Make sure certain special characters are not in names.
- Make sure all required fields are complete.
- Make sure all special graphics are imported before importing the model.

Duplicate Names (2.2.1 and 2.2.2)

When checking the names, be concerned with names that are at the same level of hierarchy. First, make sure no Activities or Processes have the same name (this should have already been enforced by SIMPROCESS 2.2.*). Second, make sure no Connectors or Pads have the same name as another Connector or Activity/Process. Pads cannot have duplicate names on the same Activity or Process, but they can have duplicate names across Activities/Processes. A Pad cannot have the same name as an Activity or Connector. Finally, make sure everything (Activity, Process, Connector, or Pad) has a name. Version 2.2.2 and earlier will allow you to have Connectors without names. Notice in the example below there are duplicate Connector names. One set of these names will need to be changed for the model to import correctly. A simple solution would be to add a 1 to each Connector name on the right. Note that once the model has been imported the names can be changed back to the original.



The next example shows a Process with a Pad that has the same name. The Process *Send Queue* has a Pad named *Send Queue*. This will cause errors during import.

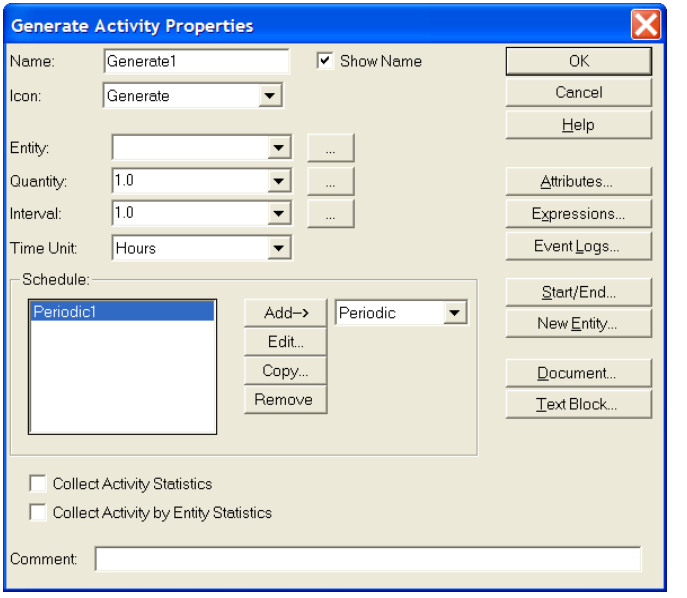


Special Characters

Names of Activities, Processes, Connectors, or Pads cannot have single quotes. These characters will cause errors in the XML parser. For instance, the name *Can't Process* will cause errors. Other characters to avoid are <, >, and /. In general, spaces are not a problem. However, extra spaces will be ignored. Therefore, *Process_1* (one space between *Process* and *1*) is the same as *Process__1* (two or more spaces between *Process* and *1*).

Required Fields

Some Activities have required fields. For instance the Generate Activity needs an entity to generate. Notice the Generate Properties Dialog below. If no entities have been defined in the model, and, thus, the Entity field in the Generate is blank, errors will occur. This same type of error could be caused by Assemble, Batch, and Transform Activities not having an entity type selected. Also, a Branch Activity with no attribute selected when branching by attribute has been set could cause errors during import.



Graphics

If bitmaps were imported for the model, `gif`, `jpeg`, or `png` versions of these bitmaps must be imported before importing the model.

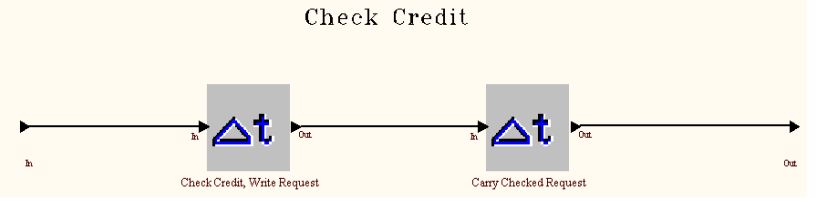
Graphical Import Results

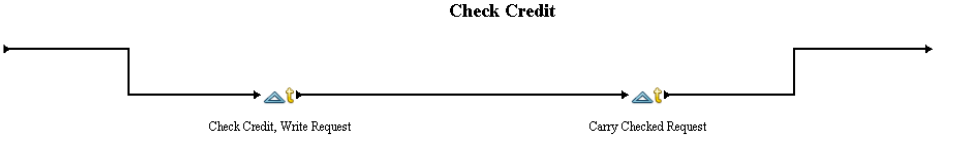
If there are no errors, all properties will import. However, the screens will look different. This is because version 3.0 or higher uses a different coordinate system from earlier SIMPROCESS versions. There are several areas that will be quite noticeably different.

- Activity/Process location
- Activity/Process size
- Connectors not connected to Pads
- Specially drawn Connectors not correct
- Text Blocks, Static labels, and Dynamic labels smaller

Activity/Process Location

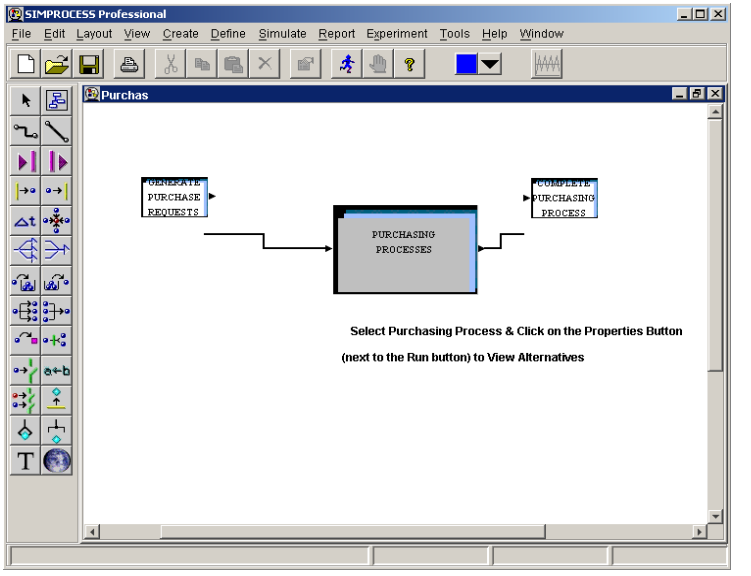
The drawing area for earlier versions of SIMPROCESS was a square. The drawing area for version 3 is a rectangle and is related to your screen size (typically 1024 x 768). Thus, Activities and Processes will be in positions relative to where they were in 2.2.1 or 2.2.2. The change from a square area to a rectangular area will cause some Activities or Processes to be out of place. The two images below demonstrate this. The first is from a 2.2.2 version model. The second is after importing to version 4. Notice that the two delay Activities may be higher or lower in relation to the Process Pads in the imported model.

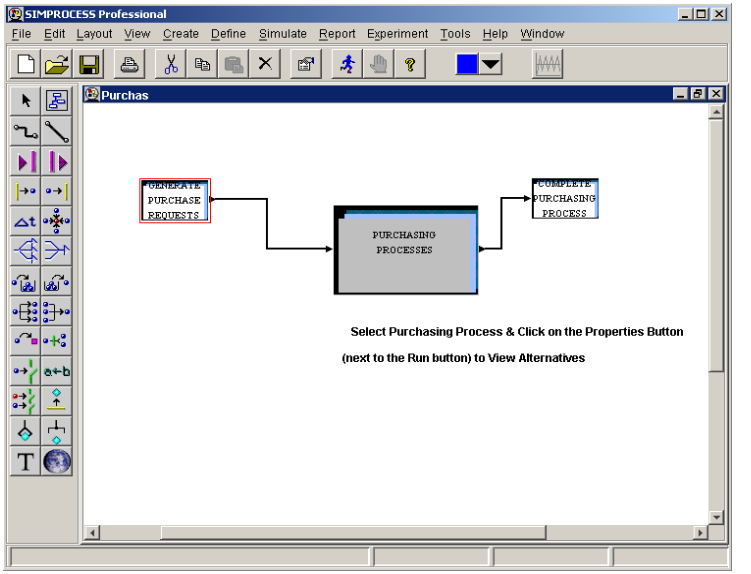




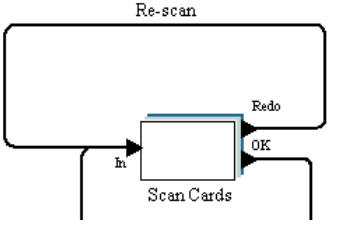
Connectors And Pads

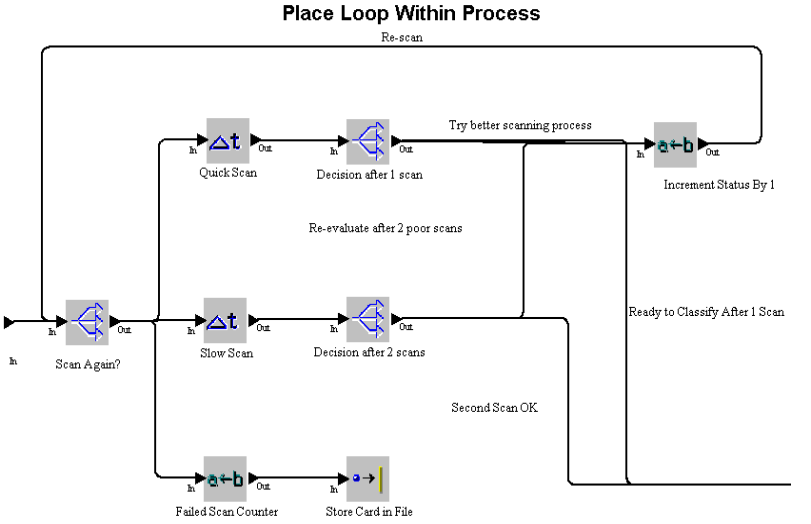
Connectors may not be attached to the Pads. This is due to the changing size of the icons. This is easily fixed by slightly moving one of the Activities or Processes. The images below demonstrate this. The first shows the model as imported. Notice that the Connectors do not connect with the Pads. The second image shows that, after a slight movement of one of the Processes, the Connectors jump into place.


SIMPROCESS Professional
File Edit Layout View Create Define Simulate Report Experiment Tools Help Window
Purchas
GENERATE PURCHASE REQUESTS
PURCHASING PROCESSES
COMPLETE PURCHASING PROCESS
Select Purchasing Process & Click on the Properties Button
(next to the Run button) to View Alternatives

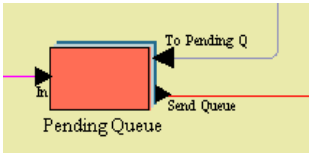

SIMPROCESS Professional
File Edit Layout View Create Define Simulate Report Experiment Tools Help Window
Purchas
GENERATE PURCHASE REQUESTS
PURCHASING PROCESSES
COMPLETE PURCHASING PROCESS
Select Purchasing Process & Click on the Properties Button
(next to the Run button) to View Alternatives

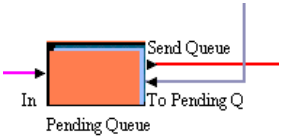
Another issue is a change in Pad rules for version 3 and higher. A Connector cannot go from the output Pad of an Activity or Process to an input Pad of the same Activity or Process. This rule is in place to help prevent infinite loops. Thus, if a 2.2.2 model had a Connector from an output Pad of a Process to an input Pad of the same Process, that Connector will not be in the imported model. This is easily fixed by placing another Activity (like a Merge) between the two, or having the loop occur within the Process. In the example below the *Re-scan* Connector in the 2.2.2 model is not allowed in 3.0 or higher. The solution is to have the loop within the Process.





Also, Pads may be arranged differently. Pads will be on the correct side of an Activity or Process, but they may be in a different order. Notice that the Pads on the right side of the Process *Pending Queue* are swapped. The first image is from the 2.2.2 model, and the second is from the imported model. Pads on other Activities like the Clone, Split, Assemble, and Gate may be swapped as well.

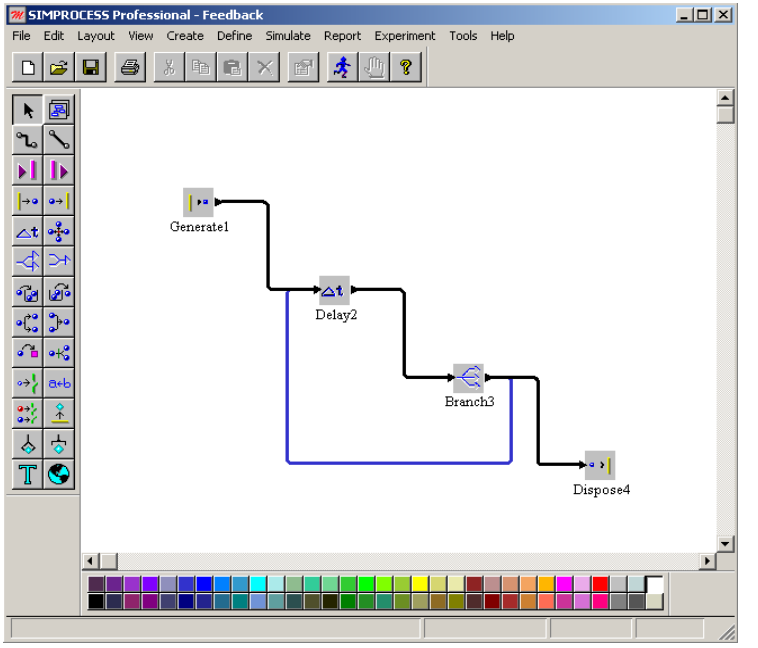




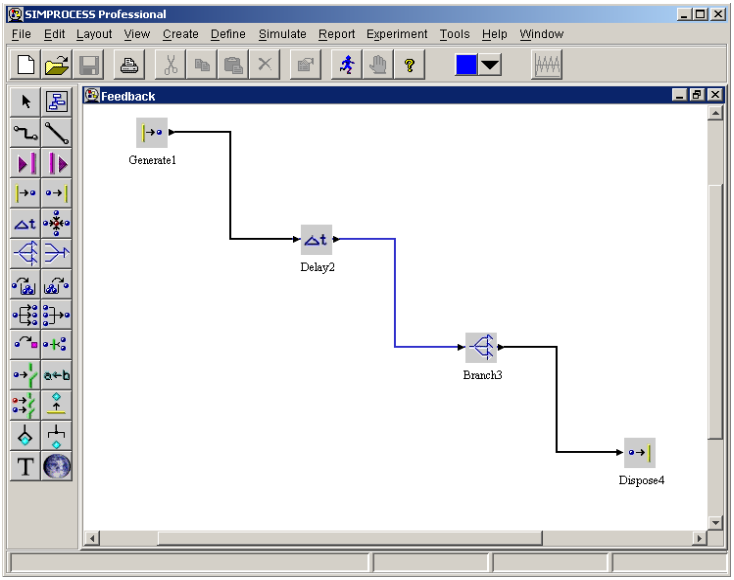
Specially Drawn Connectors

Connectors are drawn by clicking on the output Pad of one Activity and then clicking on the input Pad of another Activity. These Connectors will default to a bent Connector. Special paths for

Connectors are created by clicking on the layout. Every click point creates a right angle. In the 2.2.2 example below, this is how the feedback loop is created (blue Connector).

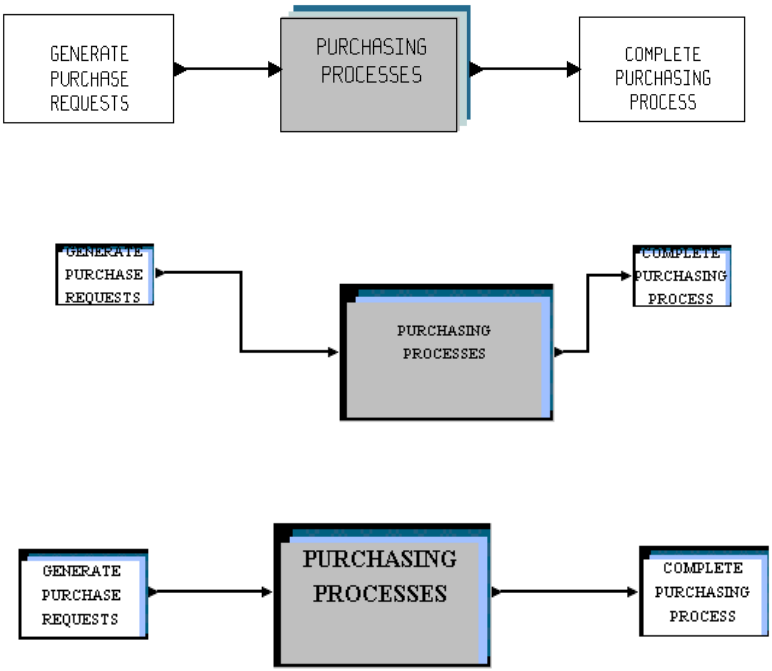


Notice that in the imported model the feedback Connector from Branch3 to Delay2 overlays the Connector from Delay2 to Branch3. This is because the Connector from Branch3 to Delay2 took the default path. Thus, this Connector will need to be deleted and redrawn. Converting Connectors using the vertices of the corners created strange results due to the change in coordinate systems. Using the default Connector between Activities means only special Connectors need to be redrawn.

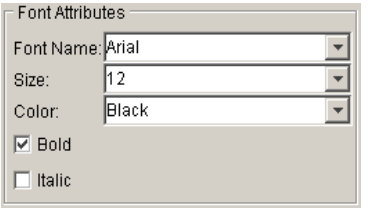


Text Blocks, Static Labels, and Dynamic Labels

Text blocks are labels that can display on Processes or Activities. In earlier versions of SIMPROCESS, the font used for text blocks was a vector font. The font would scale as the Process or Activity was resized. In SIMPROCESS 3 and higher, the text blocks were improved to allow the font attributes (font type, color, size, and style) to be set. This requires using the fonts native to your system. Therefore, the font will not scale as a Process or Activity is resized. During conversion, the default font attributes are used. The default font attributes may not fit the Process or Activity, or the default font attributes may be too small for the Process or Activity. The model below is a 2.2.2 model. The image that follows shows the imported model. Notice that for the outer two Processes the text blocks are too large and for the Process in the middle the text block is too small. Also, notice the location difference of the center Process. For the Processes where the text block is too large, either the Process can be sized larger or the text block can be sized smaller. The third image shows the corrections. The two outer Processes were sized larger. The font size was set larger for the text block for the center Process, and the font style was set to bold. Finally, the **Align** option on the **Edit** menu was used to line up the Processes.

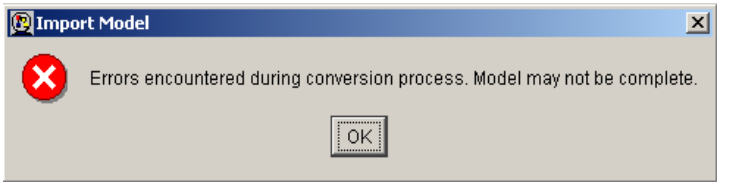


Static and dynamic labels can experience the same results during conversion. Although, labels do not overlay a Process or Activity like a text block, the imported labels may be smaller than desired. Again, this is due to vector fonts being eliminated. To change a label, just open up the properties and change the font attributes.



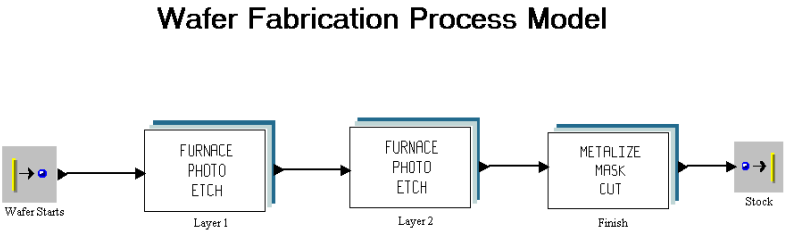
Properties Import Results

If the model has been prepared properly, there should be no errors during the import. If errors do occur, an error dialog will appear.

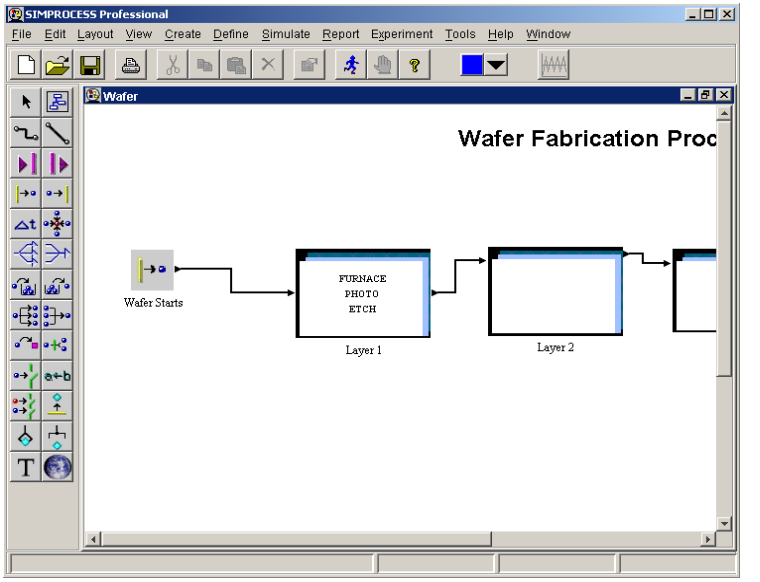


If the error dialog appears, open the `simprocess.log` file in the SIMPROCESS working directory (see "SIMPROCESS Working Directory"). This file will contain more detailed error messages. The first error listed is the one to focus on. Normally, the error messages that follow the first one result from the first error.

The image below shows the top level of the 2.2.2 demo model Wafer.spm. This model will be used as an example for diagnosing errors in properties.

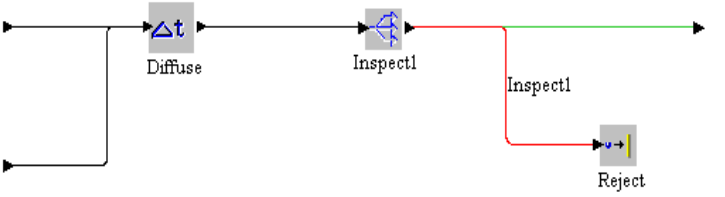


As an example, an import of the 2.2.2 demo model Wafer.spm produced the following error in simprocess.log: **Error in Activity properties of Activity Inspect1 in imported model Wafer**. This error means the problem occurred somewhere close to the Activity named Inspect1. In the image below, notice the graphics of the imported model. The layout shows that the text block is missing from the Process Layer2. Also, the Pads are not in the proper place for Layer2. This indicates that the error occurred in the Process Layer1 since no properties for Layer2 were imported.



To solve this, open the model in SIMPROCESS 2.2.2. or 2.2.1. If you do not know exactly where the Activity named is, you can use the Find Activity feature of the Activity Browser to find it.

Notice that there is a Connector named *Inspect1,* as well as the Activity named *Inspect1*. This is the cause of the problem. Changing the name of the Connector or the Activity is the solution. In this particular model, this same Activity/Connector combination is in Layer2 and needs to be changed there as well. (This error caused by duplicate names will not occur with models saved in 2.2.3.)



Import Troubleshooting

The table below gives sample error messages from the simprocess.log file along with possible causes.

If the error message gives a specific Activity type (such as Assemble, Batch, etc.) there is usually something missing from the properties for that Activity. The errors listed below are the most common. The last four are not specific to a particular Activity type.

If errors persist, contact SIMPROCESS Technical Support at `simprocess@caci.com` for assistance.

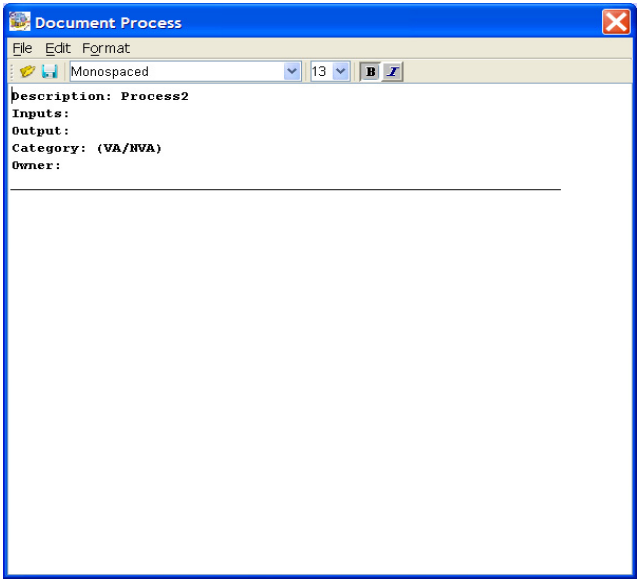
TABLE 1. Import Errors and Causes

Error	Cause
Error in schedules for Generate Generate1 in imported model Feedback	Entity field is blank in a Generate schedule
Error in Activity properties of Activity Inspect1 in imported model Wafer	A Connector or Pad has the same name as the Activity listed (2.2.1 and 2.2.2 only).
Error in Dummy Connector > in imported model Wafer	A Connector is missing a name (2.2.1 and 2.2.2 only).
SPDOM.selectSingleNode exception javax.xml.transform.TransformerException: misquoted literal... expected single quote!	A name has a single quote.
Error in Connector Rejects in imported model Wafer	Two Connectors have the same name (2.2.1 and 2.2.2 only).

Importing Document Files

Versions of SIMPROCESS prior to 3 stored model documentation in files in the model folder. These files will not import automatically. The document feature has been improved in version 3 and higher. No longer is the documentation stored in separate files. In version 3 and higher it is part of the model file. For the documentation to be part of the model, the text must be RTF. The files in earlier version models are ASCII if NotePad was used to create the text or some other format if another text editor (such as MS Word) was used. To import these files they must first be saved as RTF. This is easily done in MS Word. There is also shareware available on the internet that will convert ASCII files to RTF (`http://www.jafsoft.com/asctortf`). The names of the files converted to RTF are not important to SIMPROCESS. Once the files are RTF, simply click the **Document** button on the item for import. There is a **Document** button on all Activity, Process, entity, and resource properties dialogs. Also, there is **Model Documentation** on the **Define** menu.

Clicking the **Document** button brings up an RTF editor.



From the **File** menu choose **Read from File...** This allows you to read in the RTF document files which makes the file a part of your model file.

This must be done for every documentation file that was created for your 2.2.* model. All of these files will be located in the model's folder and will have the extension `.doc`.

APPENDIX B

Activity Summary Table

Activity	Description	Required Parameters	Non-standard Options
Assemble	Receives 2 or more entities and assembles them into a single entity for release.	Component Entity Assembled Entity	Resources Delay Batch Components
Assign	Assigns values to entity attributes.	Entity Attributes	Resources Delay Set Entity Priority
Batch	Stores entities until a condition is met, then releases them as one, while retaining their individual identity.	Quantity to Batch Min Batch Size Max Hold Time Batch Entity Type	Resources Delay
Branch	Routes entities to different paths in the model network.	Branch Type	Resources Delay

Appendix B - SIMPROCESS Activity Summary

Activity	Description	Required Parameters	Non-standard Options
Clone	Clones entities.	Number of Entities	Resources Delay
Delay	General-purpose activity.	Delay Time	Resources
Dispose	Disposes of entities.		Maximum entity Count
Free Resource	Releases resources obtained by Get Resource activities.	Release actions	Release all allocated resources
Gate	Accumulates entities until a condition is met, then releases a specified number of them.	Threshold Release ON/OFF Threshold Release Quantity (if Threshold Release ON) Trigger Release Quantity (if Trigger Release ON)	Resources Delay Batch released entities
Generate	Generates entities.	Entity type Quantity Interval	Start and End dates) Schedule Type Schedule Items
Get Resource	Obtains resources to be held during several activities.	Resources Tag attached to allocation	
Join	Reunites entities divided at a Split activity.	Family Name	Batch family members Resources Delay
Merge	Merges entities and routes them in a single stream.		
Replenish Resource	Adds units to consumable resources.	Replenish actions (Resource and Units)	
Split	Divides one entity into several to model the division of processing among several activities.	Family name	Copy Priority Resources Delay
Synchronize	Coordinates the release of various entities.	Number of Pads	Resources Delay

Appendix B - SIMPROCESS Activity Summary

Activity	Description	Required Parameters	Non-standard Options
Transfer	Transfers entities from one portion of a model to another without a connector, or transfers an entity to another SIMPROCESS model.	Send or Receive Local or Remote Connection	Most standard options are not available
Transform	Transforms arriving entities from one type to another, and releases one or more of the new type.	Number of Output Entities Output Entity type	Copy Attributes Resource Delay
Unbatch	Separates a batched entity into its constituent parts.		Resource Delay Retain Batched Entity Unbatch Nested Batches

APPENDIX C

SIMPROCESS File Structure

The SIMPROCESS installation program creates two directories called `SIMPROCESS` (unless a different name is chosen during installation). The two directories are the installation directory and the working directory. The SIMPROCESS working directory is created in the user's home directory (see "SIMPROCESS Working Directory"). (Note that on Linux systems it is possible that the installation and working directory are the same. See "SIMPROCESS Working Directory".) In these directories there are a number of important files and subdirectories. The primary subdirectories are `SPSYSTEM` (System Directory), `SPUser`, and the `models` directory. This appendix briefly describes the structures and contents of the directories. The basic directory structures are listed below. Note that the `dispatcher` directory only exists if the SIMPROCESS Dispatcher is installed. Some of these directories can be opened in the system file explorer from the **File** menu (**File/Open Directory**).

```
SIMPROCESS installation directory
   SPSYSTEM
   SPHelp
   jre
   ext
   dispatcher (optional)
   dispatchermodels (optional)
   Uninstall_SIMPROCESS


SIMPROCESS working directory
```

```
SPUser
      document
      SampleFiles
models
      Demos
      ExpressionDemos
metamodel
      BPEL
      AddressModel
      CorporateInformation
orgmodel
      Demos
dashboardserver
timeserver
```

Installation Directory

```
ReadMe.txt                ( ReadMe file with summary of changes )
SIMPROCESS.ico            ( SIMPROCESS icon - Windows only )
Dispatcher.exe            ( Optional Dispatcher application - Windows
                            only )
Dispatcher                ( Optional Dispatcher application - non-Win-
                            dows systems )
Dispatcher.lax            ( Optional Dispatcher application settings
                            file )
lax.jar                   ( Installer file )
SPRunSimulation.exe       ( SIMPROCESS with no GUI - Windows only )
SPRunSimulation           ( SIMPROCESS with no GUI - non-Windows )
SPRunSimulation.lax       ( Application settings file )
SIMPROCESS.exe            ( SIMPROCESS executable - Windows only )
SIMPROCESS                ( SIMPROCESS executable - non-Windows )
SIMPROCESS.lax            ( Application settings file )
MetamodelEditor.exe       ( Metamodel Editor executable - Windows only )
MetamodelEditor           ( Metamodel Editor executable - non-Windows )
MetamodelEditor.lax       ( Application settings file )
MetamodelEditor.ico       ( Metamodel Editor icon - Windows only )
OrgModelEditor.exe        ( OrgModel Editor executable - Windows only )
OrgModelEditor            ( OrgModel Editor executable - non-Windows )
OrgModelEditor.lax        ( Application settings file )
OrgModelEditor.ico        ( OrgModel Editor icon - Windows only )
java2rei.dll              ( Rose Java interface library - Windows only )
lmutil.exe                ( Licensing file - Windows only )
lmutil                    ( Licensing file - non-Windows )
```

```
hostid.txt                  ( Licensing file )
hostid.bat                  ( Licensing file - Windows only )
hostid.sh                   ( Licensing file - non-Windows )
License.htm                 ( SIMPROCESS License agreement )
Import Procedures.pdf       ( Documentation for importing 2.2.1 models )
```

SPSYSTEM Directory

```
InstallSettings.win.xml   ( defaults for some preference settings - Win-
dows only )
InstallSettings.lnx.xml   ( defaults for some preference settings - Linux
only )
commons-beanutils.jar     ( SIMPROCESS executable jar file )
commons-collections.jar   ( SIMPROCESS executable jar file )
commons-digester.jar      ( SIMPROCESS executable jar file )
commons-io.jar            ( SIMPROCESS executable jar file )
commons-logging.jar       ( SIMPROCESS executable jar file )
dom4j.jar                 ( SIMPROCESS executable jar file )
iText.jar                 ( SIMPROCESS executable jar file )
jasperreports.jar         ( SIMPROCESS executable jar file )
java-image-scaling.jar    ( SIMPROCESS executable jar file )
java2rei.jar              ( SIMPROCESS executable jar file - Windows only)
jcalendar.jar             ( SIMPROCESS executable jar file )
jcommon.jar               ( SIMPROCESS executable jar file )
jdt-compiler.jar          ( SIMPROCESS executable jar file )
jfreechart.jar            ( SIMPROCESS executable jar file )
jgraph.jar                ( SIMPROCESS executable jar file )
jh.jar                    ( SIMPROCESS executable jar file )
metamodel.jar             ( SIMPROCESS executable jar file )
modelfit.jar              ( SIMPROCESS executable jar file )
mpxj.jar                  ( SIMPROCESS executable jar file )
OptQuest.jar              ( SIMPROCESS executable jar file )
plot.jar                  ( SIMPROCESS executable jar file )
poi.jar                   ( SIMPROCESS executable jar file )
poi-ooxml.jar             ( SIMPROCESS executable jar file )
poi-ooxml-schemas.jar     ( SIMPROCESS executable jar file )
poi-scratchpad.jar        ( SIMPROCESS executable jar file )
simprocess.jar            ( SIMPROCESS executable jar file )
simext.jar                ( SIMPROCESS executable jar file )
SPRemote.jar              ( SIMPROCESS executable jar file )
webservices-extra.jar     ( SIMPROCESS executable jar file )
webservices-extra-api.jar ( SIMPROCESS executable jar file )
webservices-rt.jar        ( SIMPROCESS executable jar file )
webservices-tools.jar     ( SIMPROCESS executable jar file )
xalan-j.jar               ( SIMPROCESS executable jar file )
xbean.jar                 ( SIMPROCESS executable jar file )
GetStart.pdf              ( documentation file )
```

```
SPUserA.pdf                ( documentation file )
SPUserB.pdf                ( documentation file )
SPUserC.pdf                ( documentation file )
SPUserAppendices.pdf       ( documentation file )
Metamodel.pdf              ( documentation file )
OrgModel.pdf               ( documentation file )
Release.jpg                ( Splash image file )
RoseExists.dll             ( SIMPROCESS Rose library - Windows only )
license.dat                ( license file obtained from CACI )
```

SPHelp Directory

This directory contains the directories and files for the SIMPROCESS Help.

jre Directory

This directory contains the Java Runtime Environment referenced by the SIMPROCESS program.

ext Directory

This directory is a secondary location for adding external Java classes to SIMPROCESS. The preferred location is the `classes` directory within a model's directory (see "Model-Specific Java Additions" and "Method ExternalCall" for more information).

dispatcher Directory

This directory is only created if SIMPROCESS and Dispatcher is selected during installation. The files are for using SIMPROCESS with a Web service. See `SIMPROCESS Dispatcher.pdf` for more information.

dispatchermodels Directory

This directory is only created if SIMPROCESS and Dispatcher is selected during installation. Model files that are to be opened and run through the SIMPROCESS Dispatcher must be located in this directory.

Uninstall_SIMPROCESS Directory

Contains the uninstaller and related files.

Working Directory

Directory for files and folders requiring write permission. See "SIMPROCESS Working Directory".

```
UserPreferences.xml      ( created on first run, user preferences stored
here )
UserFiles.jar            ( created when importing application graphics and
```

```
icons, or when saving Libraries )
Experiments.xml          (created when experiments are defined or imported)
All .log and .err files
```

SPUser Directory

```
sProcDB.properties       ( database connection properties - required to use
the SIMPROCESS database )
mysql.sProcDB.properties( sample database connection properties for use
with a MySQL database; may serve as a template for other database tools )
SimProcDB.mdb            ( MS Access database - Windows only )
simprocessdb.sql         ( sample DDL statements to create a "simprocess"
database in MySQL for storing Experiment results; may serve as a template
for other database tools )
```

document Subdirectory

This directory is used to customize headings for Activity, Entity, Connector, Resource, and model documentation (**Document** button on properties dialogs). The directory is empty initially. The headings are customized with the use of text files that have the extension `.txt`. Each line of a file is considered a heading. For instance, the lines

```
Activity Name:
Resources Used:
```

in a file would be considered two separate headings. The contents of the `.txt` files completely replace the default headings.

Listed below are the names that can be used for the `.txt` files.

- `Activity.txt` - changes the headings for all Activities
- `process.txt` - changes the headings for processes
- `Entity.txt` - changes the headings for Entity types
- `Resource.txt` - changes the headings for Resources
- `Connector.txt` - changes the headings for Connectors
- `model.txt` - changes the headings for the **Model Description...** (**Define** menu).

The file `Activity.txt` changes the headings for all Activities. To change the documentation headings for a specific Activity, name the file the same name as the type of the Activity along with the `.txt` extension. Some examples are `delay.txt`, `replenishResource.txt`, `assemble.txt`, and `getResource.txt`. Notice that the type of the Activity must be spelled out completely with no spaces in the file name.

Not all files are required. If only the document headings for Entity types need to be changed, then `Entity.txt` is the only file that needs to exist.

There are sample files in the `SampleFiles/document` directory.

SampleFiles Subdirectory

This directory contains the source code for the files in the `com.caci.demo` package in `SPRemote.jar`. Also, sample batch files and scripts that start the Java RMI Registry, SPServer, and SPPlotServer are included, which can be copied and used as templates if desired. There is a `documents` subdirectory that contains sample files for modifying document headings.

document Subdirectory

Contains sample files for documentation headings.

models Directory

SIMPROCESS defaults to this directory as its starting point when saving or opening models.

Demos Subdirectory

This directory contains demonstration and reference models. See Chapter 5 of the *SIMPROCESS Getting Started Manual* (Demonstration and Reference Models) for a discussion of each model. It also contains a sample Dashboard file (see "SIMPROCESS Dashboards") and a sample SIMPROCESS bundle (see "Model Bundles") that contains a model that demonstrates SIMPROCESS Input Sources (see "Input Sources").

ExpressionDemos Subdirectory

This directory contains models that demonstrate some advanced features of SIMPROCESS available through the SIMPROCESS Expression Language. `SIMPROCESS Expression Demo Models.pdf`, which is located in this directory, has a brief explanation of each model.

metamodel Directory

Directory containing SIMPROCESS metamodels. Metamodels must be saved to or opened from this directory or a subdirectory of this directory.

BPEL Subdirectory

Contains collection of Business Process Execution Language (BPEL) metamodels that can be used to include BPEL metadata in a model.

AddressModel Subdirectory

Contains supporting data for AddressModel metamodel.

CorporateInformation Subdirectory

Contains supporting data for CorporateInformation metamodel.

orgmodel Directory

Directory containing SIMPROCESS Organization and Resource Models (OrgModels). OrgModels must be saved to or opened from this directory or a subdirectory of this directory.

Demos Subdirectory

Contains two demonstration OrgModels.

dashboardserver Directory

This directory contains the files required to run a Dashboard Server. (See "Displaying Dashboards".)

timeserver Directory

This directory contains the files required to run the SIMPROCESS TimeServer. See "Time Server" and `SIMPROCESS TimeServer.pdf` in this directory for more information on the TimeServer.

APPENDIX D

Statistical Distributions

The following pages give a brief description of the standard statistical distributions available in SIMPROCESS, as well as their required parameters. Some of the distributions also have a version from the Apache Commons Math library (see commons.apache.org/proper/commons-math/). These are listed below along with their SIMPROCESS designation in parentheses.

- Beta (BetACM)
- Binomial (BinACM)
- Exponential (ExpACM)
- Gamma (GamACM)
- Lognormal (LogACM)
- Normal (NorACM)
- Poisson (PoiACM)
- Triangular (TriACM)
- Uniform Integer (IntACM)
- Uniform Real (UniACM)
- Weibull (WeiACM)

Most of the parameters for continuous distributions can be classified, on the basis of their physical or geometric interpretation, as being one of three basic types: location, scale, or shape parameters. The following discussion is taken from the book *Simulation Modeling and Analysis* (Third Edition) by Law and Kelton (2000).

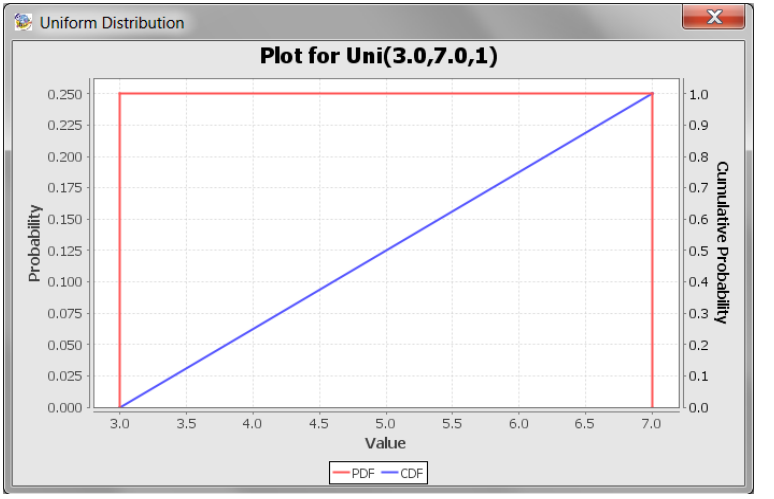
A location parameter *l* specifies an abscissa (*x* axis) location point of a distribution's range of values; usually, *l* is the midpoint (e.g., the mean of a normal distribution) or lower endpoint (e.g., location for a Pearson type V distribution) of the distribution's range. As *l* changes, the associated distribution merely shifts left or right without otherwise changing. Also, if the distribution of the random variable X has a location parameter of 0, then the distribution of the random variable $Y = X + l$ has a location parameter of *l*.

A scale parameter *b* determines the scale (or unit) of measurement of the values in the range of the distribution. (The standard deviation is a scale parameter for the normal distribution.) A change in *b* compresses or expands the associated distribution without altering its basic form. Also, if the random variable X has a scale parameter of 1, the distribution of the random variable $Y = bX$ has a scale parameter of *b*.

A shape parameter *a* determines, distinct from location and scale, the basic form or shape of a distribution with the general family of distributions of interest. A change in *a* generally alters a distribution's properties (e.g., skewness) more fundamentally than a change in location or scale. Some distributions (e.g., exponential and normal) do not have a shape parameter, while others (e.g., beta and Pearson type VI) may have two.

The last parameter of each distribution is the stream number.This is the random-number stream that will be used to general random values from the distribution. Possible values for stream are 1, 2, …, 215, with 1 being the default.

Uniform Distribution

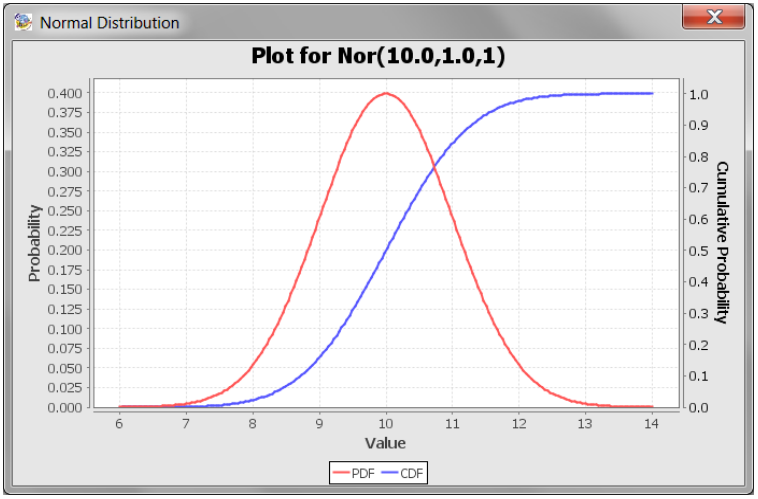


Probability density function and cumulative density function for a uniform distribution with minimum = 3 and maximum = 7.

Syntax: Uni(minimum, maximum, stream) or UniACM(minimum, maximum, stream)

The uniform distribution (continuous) is equally likely to take on any real number in the finite interval [**minimum**, **maximum**] (**minimum** >= 0.0 and **maximum** > **minimum**). The real numbers produced by a random-number generator (appear to) have a uniform distribution on the interval [0, 1].

Normal Distribution



Probability density function and cumulative density function for normal distribution function with mean = 10, standard deviation = 1. There are two normal distributions in SIMPROCESS. One only returns non-negative values (zero or higher), and the other will return negative values. For both it is required that **standard deviation** > 0.0.

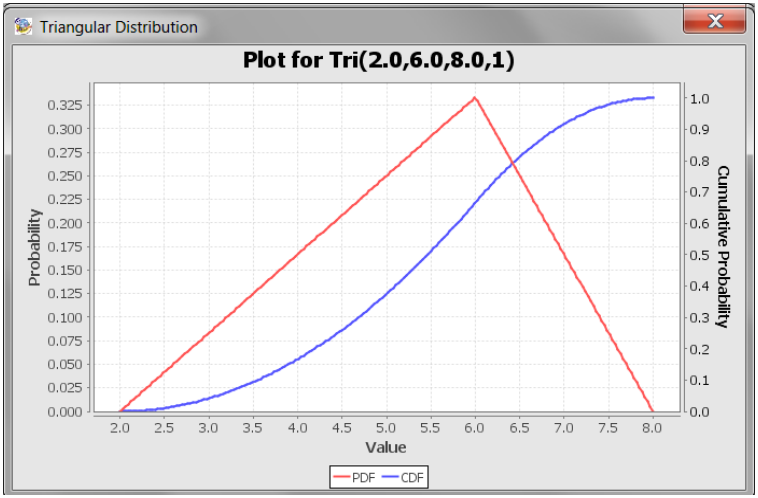
Syntax: Nor(mean, standard deviation, stream) or NorACM(mean, standard deviation, stream) - returns only non-negative values

This distribution is similar to the classical normal distribution, but if a negative value is generated, it is rejected and new values are generated until a non-negative value is generated. In general, this distribution will not be a good model for the time required to perform some task, since task-time distributions are almost always *skewed to the right*.

Syntax: Nrm(mean, standard deviation, stream) - unbounded

This is the classical normal distribution, which is found in most statistics books. It takes on real values between minus infinity and plus infinity. The density function is the familiar "bell-shaped" curve, which is symmetric about the **mean**. The probability that a value is between the **mean** minus 2 **standard deviations** and the **mean** plus two **standard deviations** is approximately 0.95. This distribution should not be used to model the time required to perform some task, since the normal distribution can take on negative values. Furthermore, as stated above, the distribution of the time to perform some task is almost always *skewed to the right*, rather than being symmetric.

Triangular Distribution

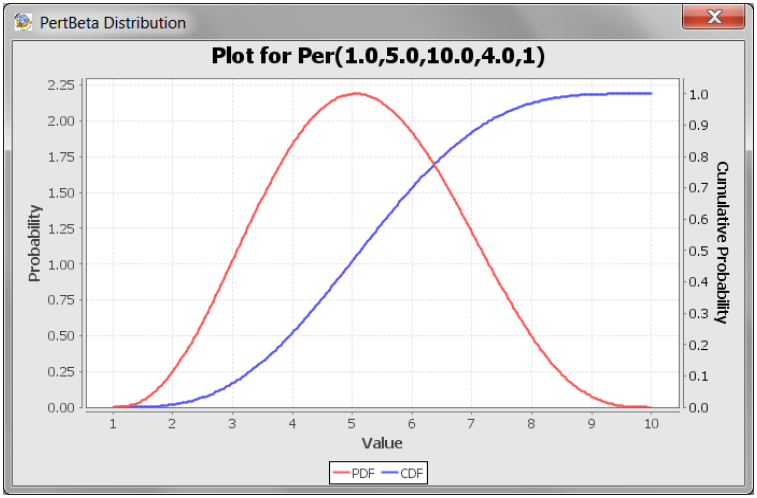


Probability density function and cumulative density function for a triangular distribution with minimum = 2, mode = 6, and maximum = 8.

Syntax: Tri(minimum, mode, maximum, stream) or TriACM(minimum, mode, maximum, stream)

The triangular distribution (continuous) is typically used as a rough model for the time required to perform some task when no real-world data are available. A triangular distribution takes on values in the finite interval [**minimum**, **maximum**] (**minimum** >= 0.0, **mode** > **minimum**, and **maximum** > **mode**), with values near the **mode** being most likely to occur. Subjective estimates of the three parameters are obtained from subject-matter experts. The mean of a triangular distribution is only equal to the **mode** when the distribution is symmetric.

PertBeta Distribution

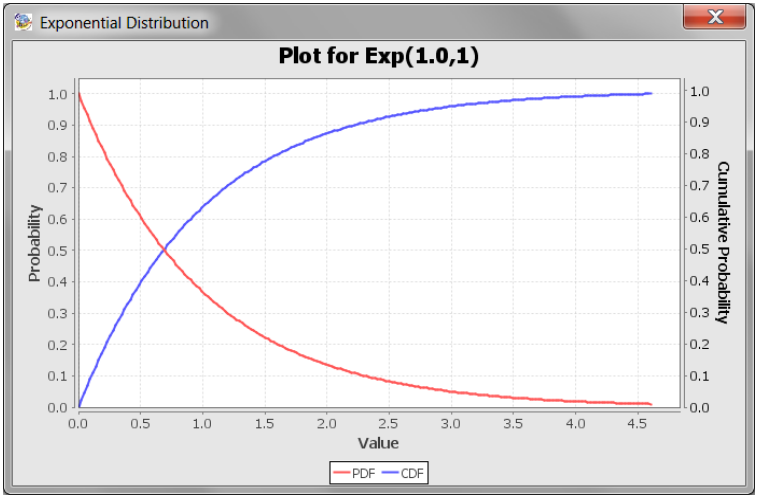


Probability density function and cumulative density function for a PertBeta distribution with minimum = 1, mode = 5, maximum = 10, and lambda = 4.

Syntax: Per(minimum, mode, maximum, lambda, stream)

The PertBeta distribution (continuous) can be used instead of the triangular distribution as a model for the time required to perform some task. The distribution produces a smooth curve and takes on values in the finite interval [**minimum**, **maximum**] (**minimum** >= 0.0, **mode** > **minimum**, and **maximum** >= **mode**), with values near the mode being most likely to occur. Subjective estimates of the three parameters are obtained from subject-matter experts. The mean of a PertBeta distribution is only equal to the mode when the distribution is symmetric. The **lambda** parameter is optional and defaults to 4.

Exponential Distribution

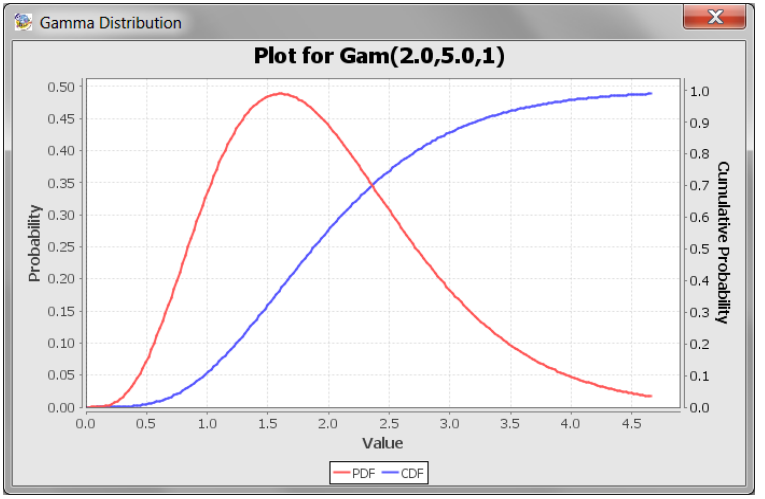


Probability density function and cumulative density function for an exponential distribution with mean = 1.

Syntax: Exp(mean, stream) or ExpACM(mean, stream)

The exponential distribution (continuous) is commonly used to model interarrival times of customers to some system when the arrival rate is approximately constant over the time period of interest. It is also sometimes used to model the time to failure of a piece of equipment. The mean of an exponential distribution is a scale parameter and must be greater than 0.0. An exponential distribution with **mean** = *m* is a gamma distribution with **mean** = *m* and **shape** = 1. An exponential distribution with **mean** = *m* is a Weibull distribution with **shape** = 1 and **scale** = *m*. If interarrival times of customers have an exponential distribution with **mean** = *m*, then the number of arrivals in any time interval of length t has a Poisson distribution (discrete) with **mean** = *t/m*.

Gamma Distribution

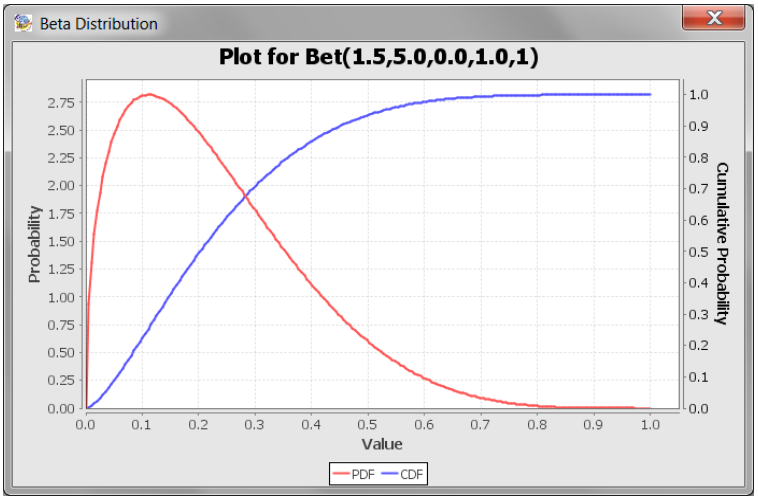


Probability density curve and cumulative density function for a gamma distribution with mean = 2 and shape = 5.

Syntax: Gam(mean, shape, stream) or GamACM(mean, shape, stream)

The gamma distribution (continuous) could be used to model the time required to perform some task. If a gamma distribution has parameters **mean** = *m* and **shape** = *a*, then *b* = *m/a* is a scale parameter. A gamma distribution with **mean** = *m* and **shape** = 1 is an exponential distribution with **mean** = *m*. When **shape** is a positive integer, the gamma distribution is an Erlang distribution. Parameter restrictions are **mean** >= 0.0 and 0.0 < **shape** < 100.0.

Beta Distribution

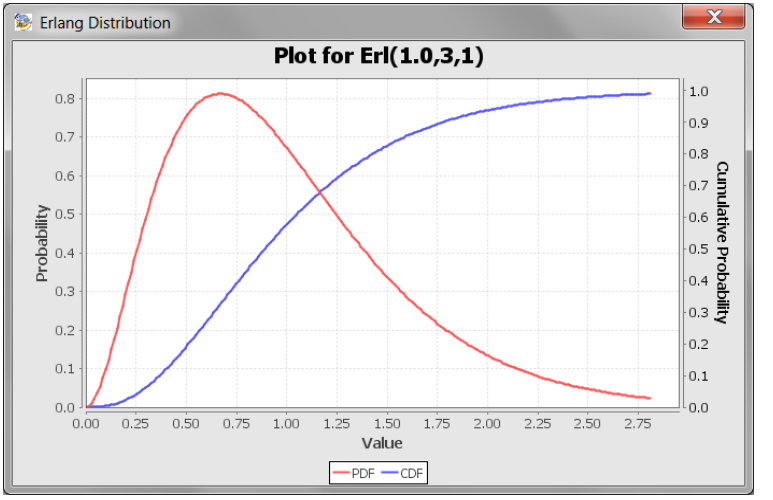


Probability density function and cumulative density function for a beta distribution with shape1 = 1.5, shape2 = 5.0, minimum = 0.0, and maximum = 1.0.

Syntax: Bet(shape1, shape2, minimum, maximum, stream) or BetACM(shape1, shape2, minimum, maximum, stream)

The beta distribution (continuous) could be used to model the time required to perform some task when the possible values are restricted to the finite interval [**minimum**, **maximum**] (**minimum** >= 0.0, **maximum** >= 1.0, and **maximum** > **minimum**). Parameter restrictions for **shape1** and **shape2** are **shape1** >= 0.0 and **shape2** > 0.0. The density function is skewed to the left, symmetric, or skewed to the right if **shape1** > **shape2**, **shape1** = **shape2**, or **shape1** < **shape2**, respectively. A beta distribution with **shape1** = **shape2** = 1 is a uniform distribution with the interval [0, 1].

Erlang Distribution

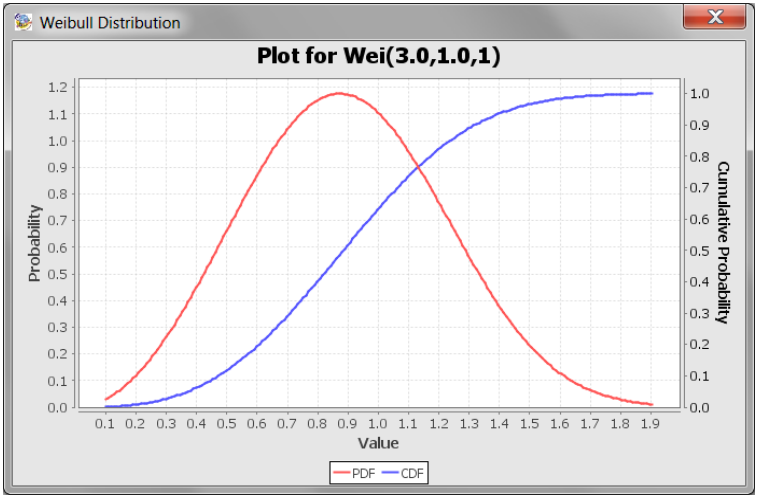


Probability density function and cumulative density function for an Erlang distribution with mean = 1 and shape = 3.

Syntax: Erl(mean, shape, stream)

The Erlang distribution (continuous) could be used to model the time required to perform some task. If an Erlang distribution has parameters **mean** = *m* and **shape** = *a*, then *b* = *m/a* is a scale parameter. An Erlang distribution is just a gamma distribution whose shape parameter is a positive integer. The sum of *k* exponential random variables with **mean** = *m* is an Erlang distribution with **mean** = *km* and **shape** = *k*. Parameter restrictions are **mean** >= 0.0 and 0.0 < **shape** < 100.0.

Weibull Distribution

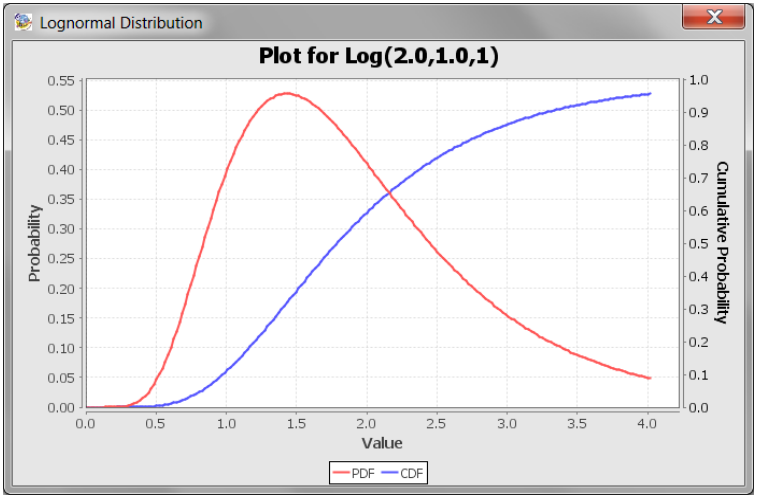


Probability density function and cumulative density function for a Weibull distribution with shape = 3 and scale = 1.

Syntax: Wei(shape, scale, stream) or WeiACM(shape, scale, stream)

The Weibull distribution (continuous) could be used to model the time required to perform some task. It is also sometimes used to model the time to failure of a piece of equipment. A Weibull distribution with parameters **shape** = 1 and **scale** = *b* is an exponential distribution with **mean** = *b*. The Weibull distribution is skewed to the left when **shape** > 3.6. Parameter restrictions are **shape** >= 0.0 and **scale** > 0.0.

Lognormal Distribution

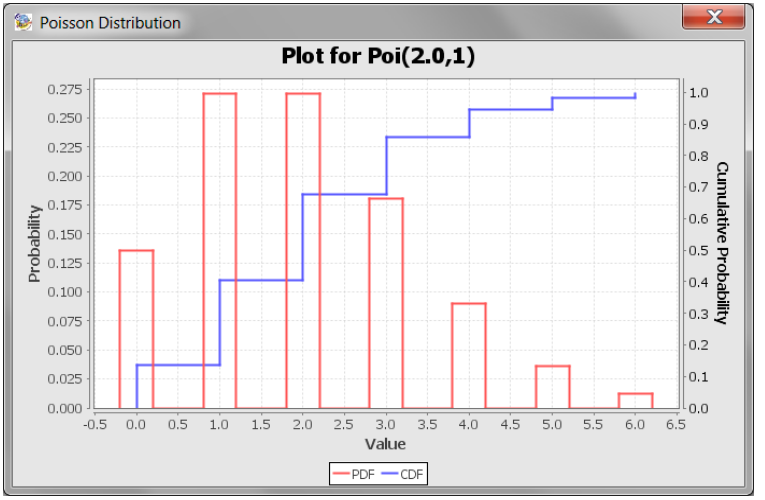


Probability density function and cumulative density function for a lognormal distribution with mean = 2 and standard deviation = 1.

Syntax: Log(mean, standard deviation, stream) or LogACM(mean, standard deviation, stream)

The lognormal distribution could be used to model the time required to perform some task when "large" values sometimes occur. It is always skewed to the right and it has a longer right tail than the gamma or Weibull distributions. The lognormal distribution is closely related to the classical normal distribution - see the book *Simulation Modeling and Analysis* (Third Edition) by Law and Kelton (2000) for details. Furthermore, the parameters of the lognormal distribution, namely, **mean** and **standard deviation**, correspond to the lognormal distribution and are *not* the mean and standard deviation of the corresponding normal distribution. Parameter restrictions are **mean** > 0.0 and **standard deviation** > 0.0.

Poisson Distribution

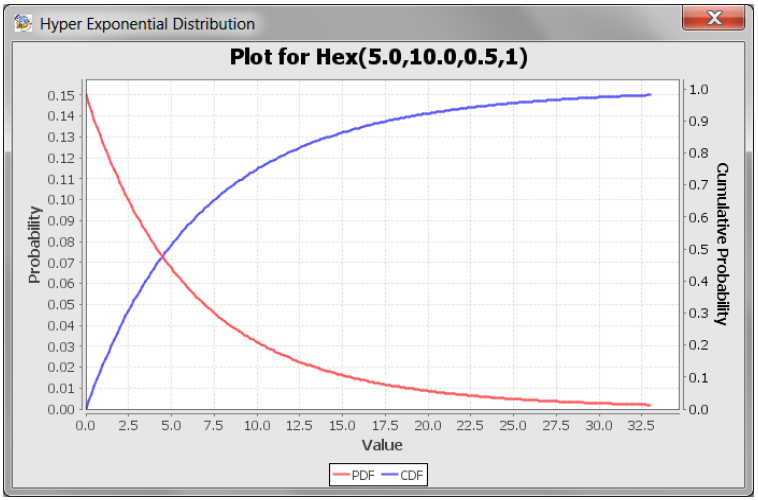


Poisson distribution with mean = 2.

Syntax: Poi(mean, stream) or PoiACM(mean, stream)

The Poisson distribution (discrete) with **mean** = *m* is the distribution of the number of customers that arrive to some system in any time interval of length 1 when the interarrival times have an exponential distribution (continuous) with **mean** = 1/*m*. **Mean** must be greater than 0.0.

Hyper Exponential Distribution

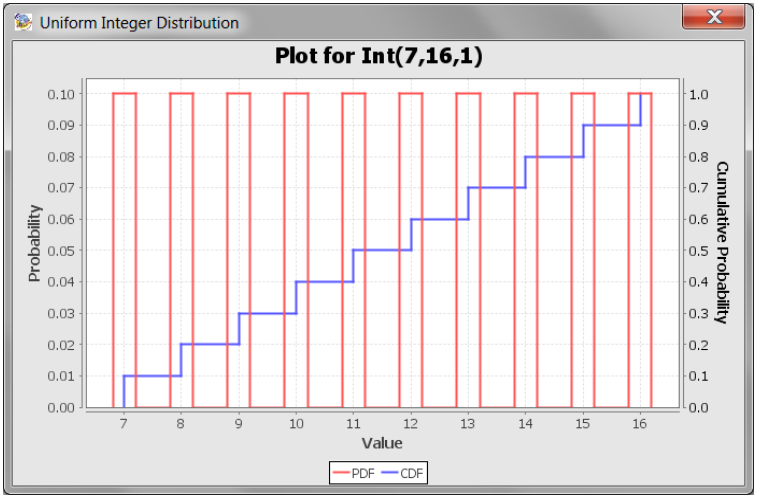


Hyper exponential distribution with mean1 = 5, mean2 = 10, and probability1 = 0.5.

Syntax: Hex(mean1, mean2, probability1, stream)

The hyper exponential distribution (continuous) is a mixture of two exponential distributions. Specifically, a hyper exponential distribution with parameters **mean1**, **mean2**, and **probability1** takes on values from an exponential distribution with parameter **mean1** with a probability of **probability1** and takes on values from an exponential distribution with parameter **mean2** with a probability of 1 - **probability1**. A hyper exponential distribution with **probability1** = 1 is an exponential distribution with parameter **mean1**. Parameter restrictions are **mean1** >= 0.0, **mean2** > 0.0, and 0.0 <= **probability1** <= 1.0.

Uniform Integer Distribution

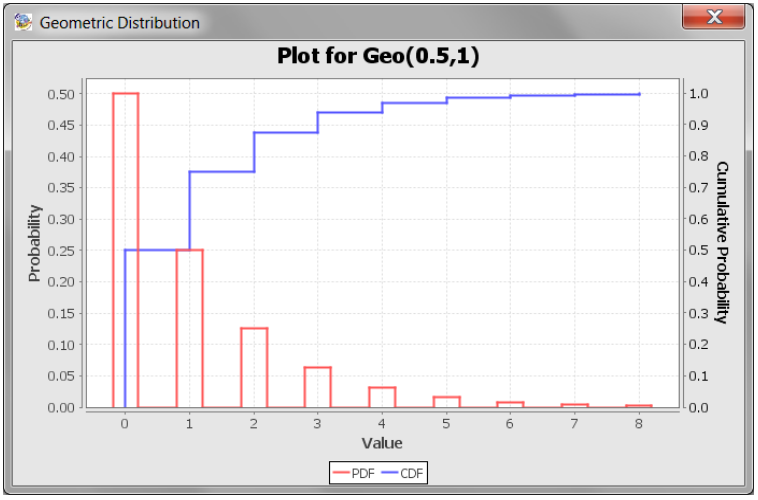


Probability density function and cumulative density function for uniform integer distribution with minimum = 7 and maximum = 16.

Syntax: Int(minimum, maximum, stream) or IntACM(minimum, maximum, stream)

A uniform integer distribution (discrete) is equally likely to take on any integer in the finite interval [**minimum**, **maximum**], where **minimum** and **maximum** are integers with **minimum** >= 0 and **minimum** < **maximum**.

Geometric Distribution

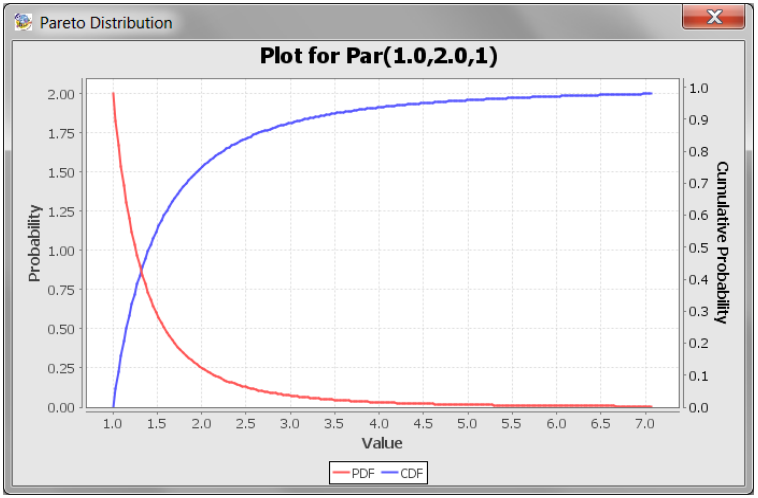


Probability density function and cumulative density function for a geometric distribution with probability = 0.5.

Syntax: Geo(probability, stream)

The geometric distribution (discrete) with **probability** = *p* can be thought of as the distribution of the number of failures before the first success in a sequence of independent Bernoulli trials, where success occurs on each trial with a probability of *p* and failure occurs on each trial with a probability of 1 - *p*.

Pareto Distribution

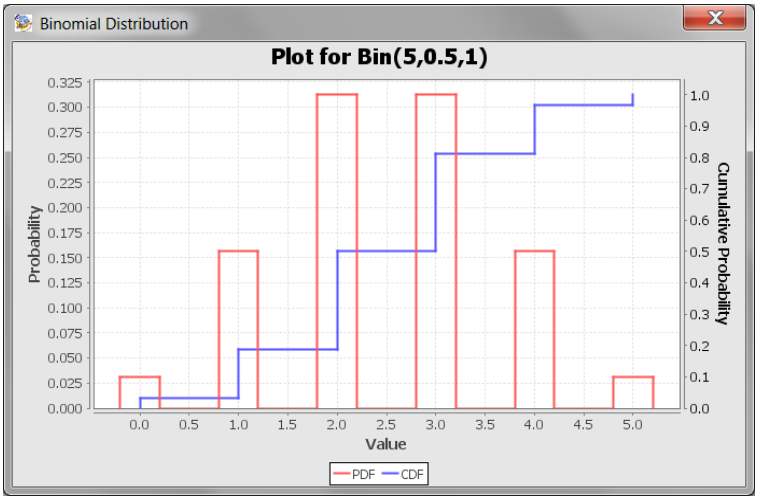


Probability density function and cumulative density function for a Pareto distribution with location = 1 and shape = 2.

Syntax: Par(location, shape, stream)

The Pareto distribution (continuous) could be used to the model interarrival times of customers (e.g., messages) when the traffic is bursty. The mean and variance are finite only if **shape** > 2. Parameter restrictions are **location** >= 0.0 and **shape** > 0.0.

Binomial Distribution

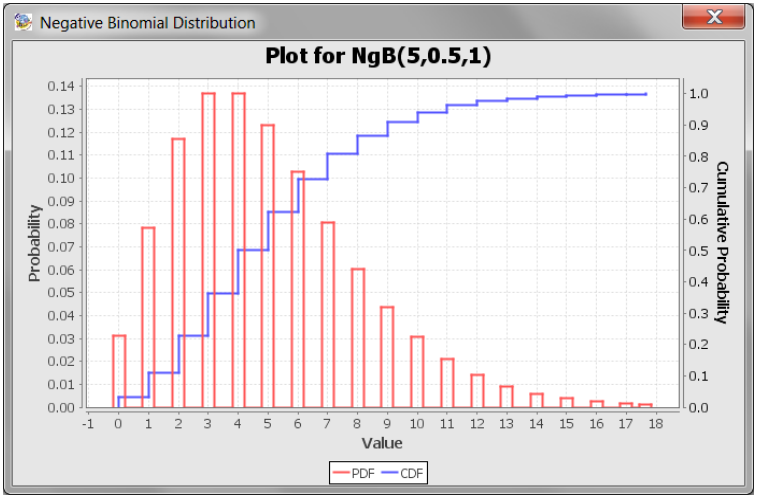


Probability density function and cumulative density function for a binomial distribution with trials = 5 and probability = 0.5.

Syntax: Bin(trials, probability, stream) or BinACM(trials, probability, stream)

The binomial distribution (discrete) with parameters **trials** = t (a positive integer) and **probability** = p can be thought of as the distribution of the number of successes in t independent Bernoulli trials, where success occurs on each trial with a probability of p and failure occurs on each trial with a probability of 1 - p. A binomial distribution with **trials** = 1 is called a Bernoulli distribution with **probability** = p.

Negative Binomial Distribution

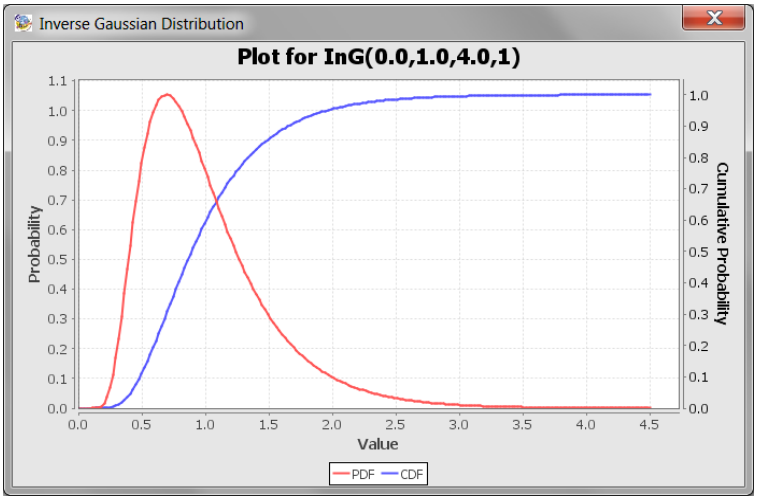


Probability density function and cumulative density function for a negative binomial distribution with s = 5 and probability = 0.5.

Syntax: NgB(s, probability, stream)

The negative binomial distribution (discrete) with parameters **s** (> 0.0) and **probability** = *p* can be thought of as the distribution of the number of failures before the **s**th success in a sequence of independent Bernoulli trials, where success occurs on each trial with a probability of *p* and failure occurs on each trial with a probability of 1 - *p*. A negative binomial distribution with parameters **s** = 1 and **probability** = *p* is a geometric distribution with **probability** = *p*.

Inverse Gaussian Distribution

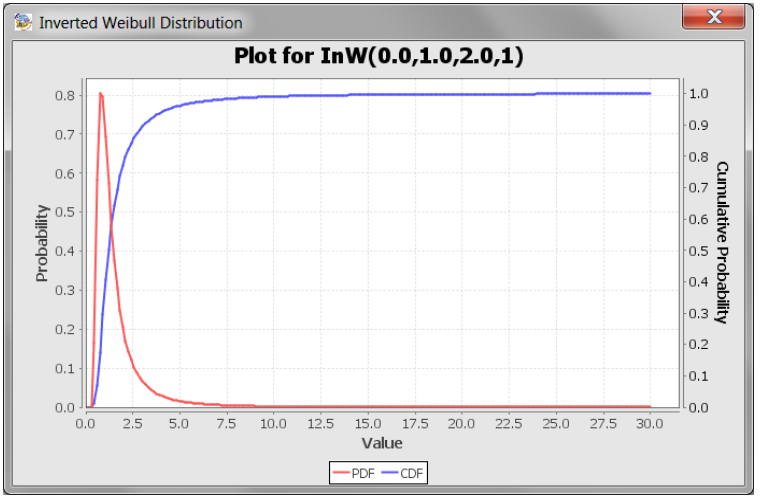


Probability density function and cumulative density function for an inverse Gaussian distribution with location = 0, scale = 1, and shape = 4.

Syntax: InG(location, scale, shape, stream)

The inverse Gaussian distribution (continuous) could be used to model the time required to perform some task. Parameter restrictions are **scale** > 0.0 and **shape** > 0.0.

Inverted Weibull Distribution

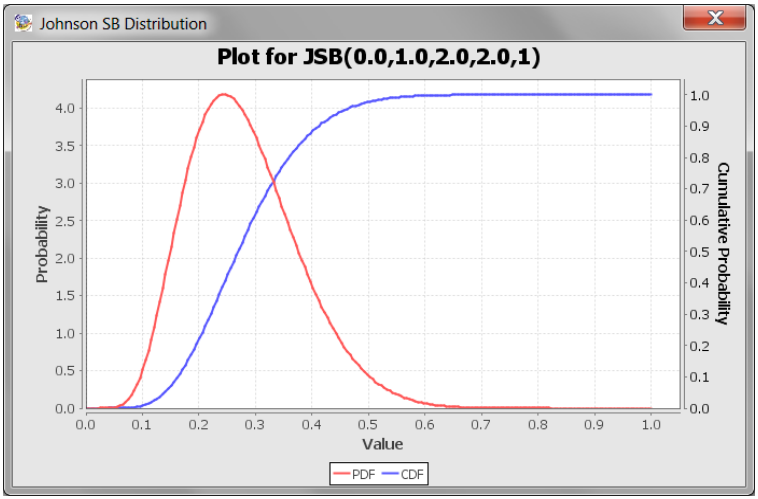


Probability density function and cumulative density function for an inverted Weibull distribution with location = 0, scale = 1, and shape = 2.

Syntax: InW(location, scale, shape, stream)

The inverted Weibull distribution (continuous) could be used to model the time required to perform some task. The mean and variance are finite only if **shape** > 2. If the random variable X has an inverted Weibull distribution with **location** = 0, **scale** = *b*, and **shape** = *a*, then Y = 1/X has a Weibull distribution with **scale** = 1/*b* and **shape** = *a*. (The **location** parameter is 0.) Parameter restrictions are **scale** > 0.0 and **shape** > 0.0.

Johnson S_B Distribution

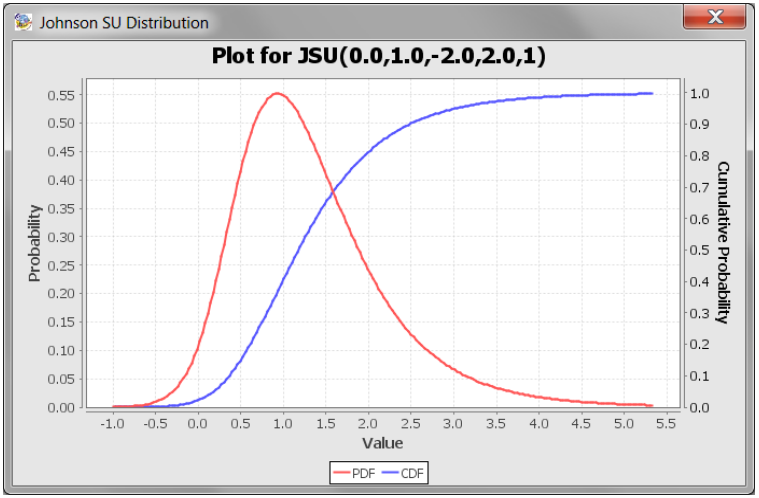


Probability density function and cumulative density function for a Johnson S_B distribution with minimum = 0, maximum = 1, shape1 = 2, and shape2 = 2.

Syntax: JSB(minimum, maximum, shape1, shape2, stream)

The Johnson S_B distribution (continuous) could be used to model the time required to perform some task when the possible values are restricted to the finite interval [**minimum**, **maximum**]. The density function is skewed to the left, symmetric, or skewed to the right if **shape1** > 0, **shape1** = 0, or **shape1** < 0, respectively. The Johnson S_B distribution is closely related to the classical normal distribution - see the book *Simulation Modeling and Analysis* (Third Edition) by Law and Kelton (2000) for details. Parameter restrictions are **shape2** > 0.0 and **maximum** > **minimum**.

Johnson S_U Distribution

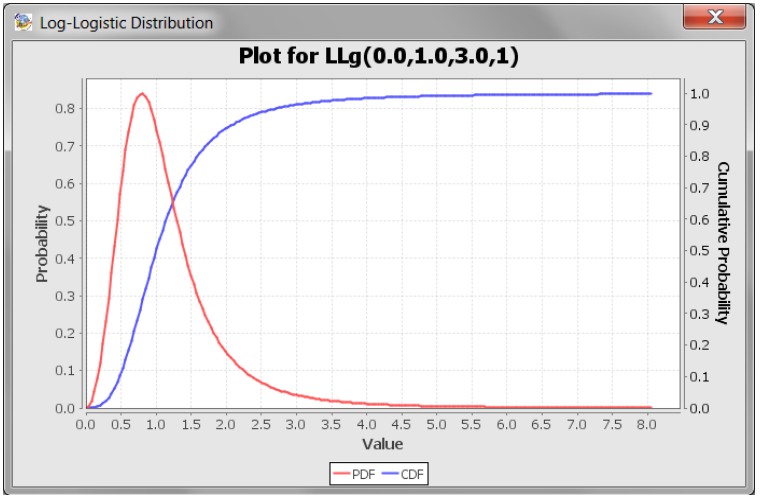


Probability density function and cumulative density function for a Johnson S_U distribution with location = 0, scale = 1, shape1 = -2, and shape2 = 2.

Syntax: JSU(location, scale, shape1, shape2, stream)

The Johnson S_U distribution (continuous) could be used to model a random variable that can take on any value between minus infinity and plus infinity. The density function is skewed to the left, symmetric, or skewed to the right if **shape1** > 0, **shape1** = 0, or **shape1** < 0, respectively. The Johnson S_U distribution is closely related to the classical normal distribution - see the book *Simulation Modeling and Analysis* (Third Edition) by Law and Kelton (2000) for details. Parameter restrictions are **scale** > 0.0 and **shape2** > 0.0.

Log-Logistic Distribution

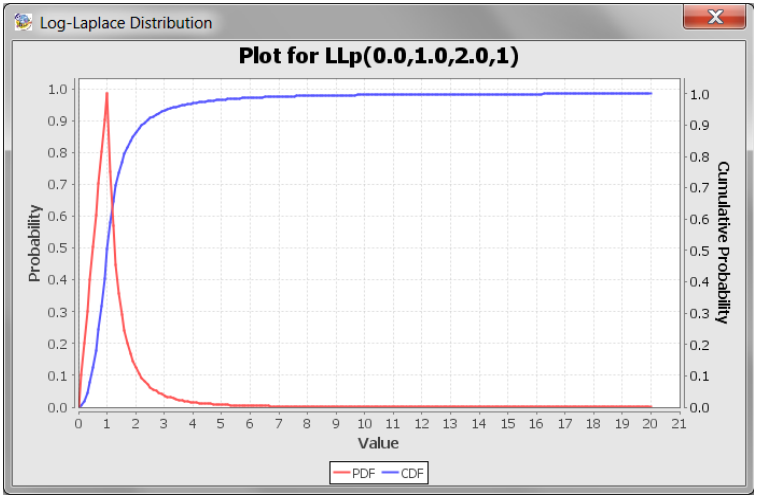


Probability density function and cumulative density function for a Log-Logistic distribution with location = 0, scale = 1, and shape = 3.

Syntax: LLg(location, scale, shape, stream)

The log-logistic distribution (continuous) could be used to model the time required to perform some task. The mean and variance are finite only if **shape** > 2. Parameter restrictions are **scale** > 0.0 and **shape** > 0.0.

Log-Laplace Distribution

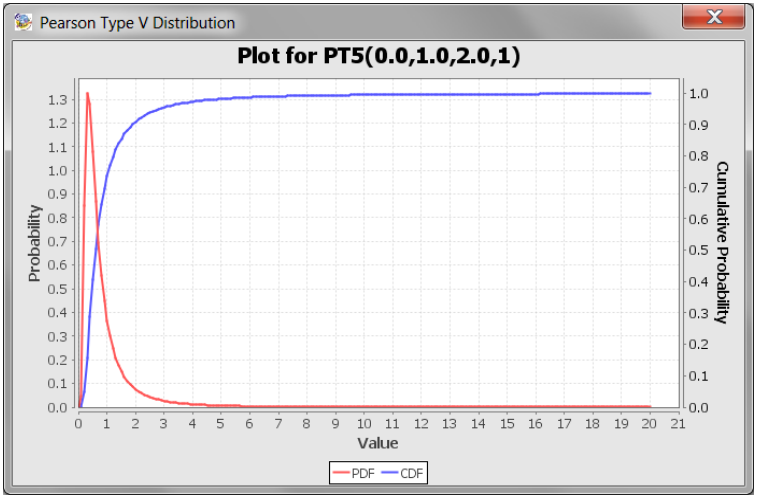


Probability density function and cumulative density function for a log-Laplace distribution with location = 0, scale = 1, and shape = 2.

Syntax: LLp(location, scale, shape, stream)

The log-Laplace distribution (continuous) could be used to model the time required to perform some task. The mean and variance are finite only if **shape** > 2. Parameter restrictions are **scale** > 0.0 and **shape** > 0.0.

Pearson Type V Distribution

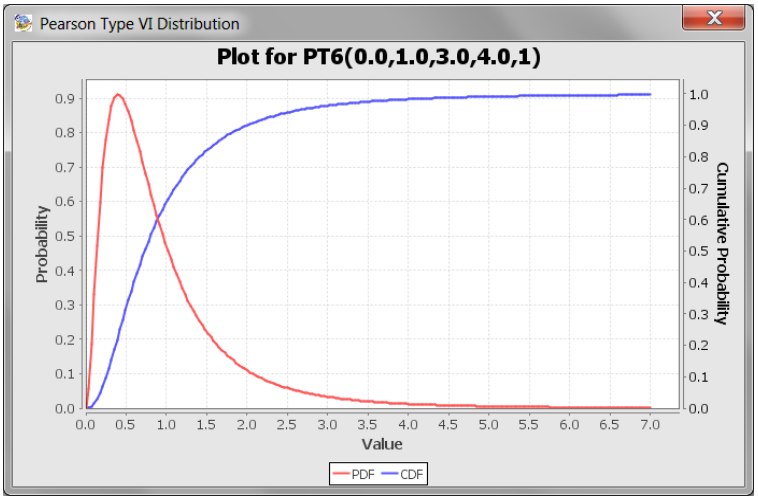


Probability density function and cumulative density function for a Pearson type V distribution with location = 0, scale = 1, and shape = 2.

Syntax: PT5(location, scale, shape, stream)

The Pearson type V distribution (continuous) could be used to model the time required to perform some task. The mean and variance are finite only if **shape** > 2. The Pearson type V distribution is closely related to the gamma distribution - see the book *Simulation Modeling and Analysis* (Third Edition) by Law and Kelton (2000) for details. Parameter restrictions are **scale** > 0.0 and **shape** > 0.0.

Pearson Type VI Distribution

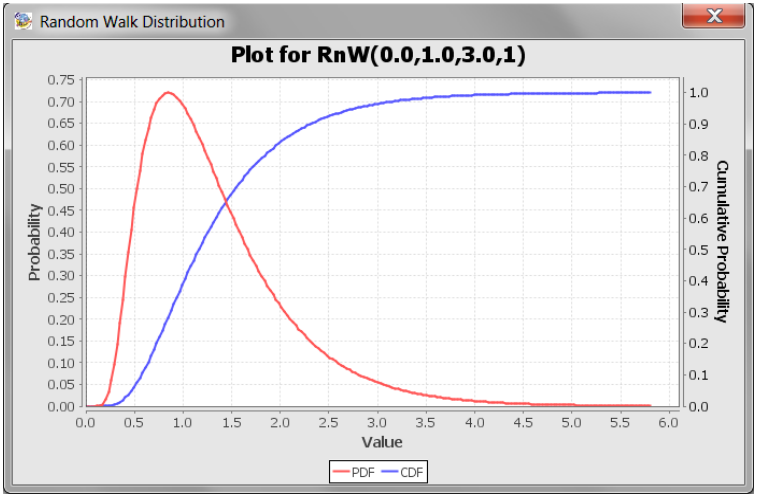


Probability density function and cumulative density function for a Pearson type VI distribution with location = 0, scale = 1, shape1 = 3, and shape2 = 4.

Syntax: PT6(location, scale, shape1, shape2, stream)

The Pearson type VI distribution (continuous) could be used to model the time required to perform some task. The density function can take on a wide variety of shapes because it has two shape parameters **shape1** and **shape2**. The mean and variance are finite only if **shape2** > 2.The Pearson type VI distribution is closely related to the beta distribution - see the book *Simulation Modeling and Analysis* (Third Edition) by Law and Kelton (2000) for details. Parameter restrictions are **scale** > 0.0 and **shape1** > 0.0.

Random Walk Distribution

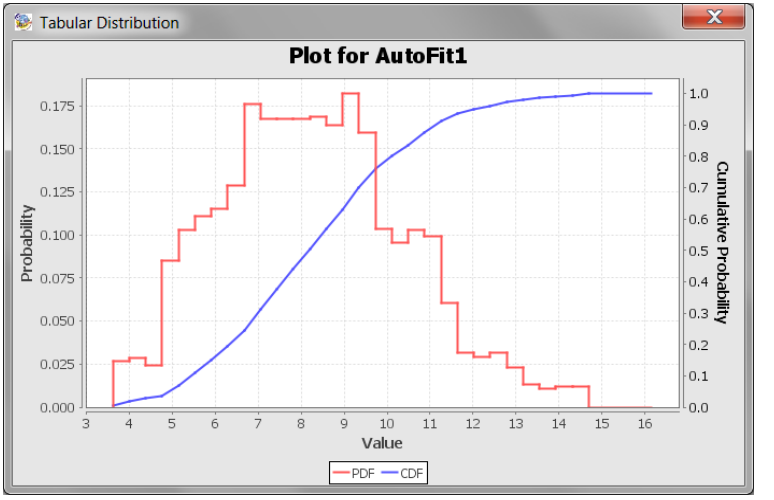


Probability density function and cumulative density function for a random walk distribution with location = 0, scale = 1, and shape = 3.

Syntax: RnW(location, scale, shape, stream)

The random walk distribution (continuous) could be used to model the time required to perform some task. Parameter restrictions are **scale** > 0.0 and **shape** > 0.0.

Empirical Distribution



Empirical distributions are not found on the SIMPROCESS distribution list. An empirical distribution can only be created by an Auto Fit distribution. (See "Auto Fits Distributions".) Thus, the name of the Auto Fit distribution appears on the SIMPROCESS distribution list, not the empirical distribution itself.

Empirical distributions consist of pairs of cumulative probability and value and can be discrete or continuous. This is the same format as Tabular Distributions. (See "Tabular Distributions".)

APPENDIX E

Statistical Tools Glossary

Absolute Deviation

The average deviation of the data set from the mean. It is computed using the absolute value of the difference between a data point and the mean, rather than squared differences used in computing the variance.

Alpha

The probability that an estimate of a parameter does not contain the true value. In statistics, this is referred to as Type I error.

Alternate Hypothesis

The complement of the null hypothesis.

Cdf

Cumulative distribution function.

χ^2 goodness of fit test

A goodness of fit test that uses an χ^2 statistic to evaluate the goodness of fit. The χ^2 goodness of fit test creates an empirical distribution for the data. The empirical distribution used is a histogram. The height of each bin of the histogram is equal to the number of points in the data set that fall between the lower and upper bounds of the bin. The χ^2 statistic is computed as the sum of squared differences of the values observed for each bin of the histogram to the number of observations expected from the probability distribution over the same range as the bin.

Confidence Interval

An interval that contains the true value of a parameter with a given probability.

Continuous Domain

A domain whose variables can take on any value in an interval (real numbers). Integer domains are sometimes interpreted as continuous domains with the implicit assumption that the integers represent an underlying continuous domain.

Data Set

A vector (or equivalently, an array or column) of data points that are the realization of a random process.

Degrees of Freedom

The number of independent elements in a statistical computation. The degrees of freedom must be known to compute the probability of a test statistic such as the χ^2 statistic.

Dof

Acronym for degrees of freedom.

Enumeration Domain

A domain whose range is the set of elements in the enumeration.

Goodness of fit test

A statistical test used to determine the probability that an observed data set came from a probability distribution. The null hypothesis is that the data set is drawn from the probability distribution. The alternate hypothesis is that the data set is not drawn from the probability distribution. A test statistic is computed to evaluate the hypothesis. If the probability of observing the test statistic is small (less than alpha), the fit is rejected.

IID

Identically and independently distributed. In simulation, identical means that observations are made from a hypothesized distribution of values that does not change over simulated time. Independent means that observations of one replication are not correlated with other observations, i.e., if one observation has a value of X, then the next observation is probably close to X.

Image Editor

An editor that contains graphs, tables, or text for viewing the results of a statistical analysis.

Integer-valued Domain

A domain whose range is the set of integer numbers.

KS

Acronym for Kolmogorov-Smirnov.

KS goodness of fit test

A goodness of fit test that uses the D statistic to evaluate the goodness of fit. The KS goodness of fit test creates an empirical distribution for the data. The empirical distribution assigns a probability of 1/N to each data point where N is number of data points in the data set. The D statistic is computed as the maximum difference between the cumulative distribution function of the empirical distribution and the cumulative distribution function of the probability distribution being tested.

Kurtosis

Characterizes the relative peakedness or flatness of a distribution relative to the normal distribution. A data set with positive kurtosis has a sharper peak than a normal distribution, while a data set with negative kurtosis has a flatter peak than the normal distribution.

Maximum Likelihood Estimate

A statistical technique for estimating the parameters of a hypothesized distribution of a data set.

Mean

Average value of a data set.

MLE

Acronym for maximum likelihood estimate.

Mode

The mode of a probability distribution is the value of X where it takes on its maximum value. If the distribution is multiply-peaked, there may not be a unique mode.

Model Element

A named object in the simulation model that has one or more simulation input variables that can be designated as factors for an experiment. The model elements include Activities, Resources, and Entity types.

Pdf

Probability density function.

Performance Measure.

An output statistic that quantifies some behavior of the model.

Percentiles

In a data set ranked by value of the data points, a percentile refers to the data point at the relative position by rank of the data point in the data set. For example, in a data set of a hundred points, the 75^{th} percentile (75%) point is the data point that is in the 75^{th} position when ranked by value.

Quartile Range

Difference between the 25^{th} percentile value and 75^{th} percentile value of the data set.

Range

Difference between the minimum and maximum values of a data set.

Real-valued Domain

A domain whose range is the set of real numbers.

Residuals

The differences between the actual values observed and the expected values as computed by a statistical test.

Sample Domain

A domain whose range is the set of probability distributions supported by SIMPROCESS.

Skewness

Characterizes the degree of asymmetry of a distribution around its mean. A data set with positive skewness has a long tail to the right of its mean. A data set with negative skewness has a long tail to the left of its mean.

Standard Deviation

Square root of the variance.

Standard Error

A measure of the spread of the data set about the mean. Typically, standard error is the one sigma error about the mean of the data set computed as the standard deviation divided by the square root of the number of data points.

Variance

Variability of data set about the mean.

APPENDIX F

SIMPROCESS System Attributes and Methods

Chapter 2 of Part B (Customizing a Model with Attributes and Expressions) discussed user defined attributes. SIMPROCESS also includes numerous predefined System Attributes and Methods. Each is listed along with a description. Also, there are examples for several of the System Methods.

System Attributes

This section lists all the SIMPROCESS system attributes in a table.

SIMPROCESS System Attributes

Model Elements	Attribute Name[a]	Get/ Set	Attribute Type	Description
System Level[b]	Model	Get	OBJECT	Reference of the Model being simulated.
	Self	Get	OBJECT	Reference of the Model Element itself.
	Background	Get	BOOLEAN	TRUE if the model is being simulated without the GUI.
	FilePath	Get	STRING	Complete file path to the model's directory including the trailing separator.
	Replication	Get	INTEGER	Current replication number.
	UNDEFINED	Get	OBJECT	Returns an undefined object.
Entity Type	Name	Get	STRING	Name of the Entity.
	Priority	Both	INTEGER	Default priority for Entity instances of this type.
	NumberIn	Get	INTEGER	Number of Entity instances of this type currently in the system.
	NumberCreated	Get	INTEGER	Number of Entity instances generated for this type.
	NumberDisposed	Get	INTEGER	Number of disposed Entity instances of this type.
	NumberWaiting	Get	INTEGER	Number of Entity instances of this type waiting for resources
	NumberOnHold	Get	INTEGER	Number of Entity instances of this type holding for condition

SIMPROCESS System Attributes

Model Elements	Attribute Name[a]	Get/ Set	Attribute Type	Description
	Interrupt	Both	BOOLEAN	Determines whether Entity instances of this type will interrupt the processing of a lower priority Entity.
Entity (instance)	Name	Get	STRING	Name of this Entity.
	Activity	Get	OBJECT	Reference of Activity this Entity instance resides in.
	AcceptTime	Get	REAL	Time Entity enters an Activity or Process.
	ActivityTime	Get	REAL	Total time Entity was in an Activity, whether waiting, on hold, or processing. Does not apply to Processes.
	BatchSize	Get	INTEGER	Number of Entity instances in this Entity's batch.
	CreateTime	Get	REAL	Time the Entity instance was created.
	EndWait	Both	BOOLEAN	TRUE if Entity wait for Resources was interrupted by MaxWait time.
	EnterTime	Get	REAL	Time the Entity instance entered the current Activity.
	HasParent	Get	BOOLEAN	TRUE if the Entity instance was created by a Split Activity.
	HoldingForCondition	Get	BOOLEAN	TRUE if the Entity is in a hold state at an Activity.
	InBatch	Get	BOOLEAN	TRUE if the Entity is part of a Batch.
	Interrupt	Both	BOOLEAN	Determines whether this Entity instance will interrupt the processing of a lower priority Entity.
	LastDelay	Get	REAL	Last processing delay time at an Activity.

SIMPROCESS System Attributes

Model Elements	Attribute Name[a]	Get/ Set	Attribute Type	Description
	MaxWait	Both	REAL	Maximum time Entity will wait for Resource allocation. Default is 0.0 (unlimited).
	NumberIn	Get	INTEGER	Number of Entity instances of the same type currently in the system.
	NumberCreated	Get	INTEGER	Number of Entity instances of the same type has generated.
	NumberDisposed	Get	INTEGER	Number of Entity instances of the same type has disposed.
	NumberWaiting	Get	INTEGER	Number of Entity instances of the same type waiting for resources
	NumberOnHold	Get	INTEGER	Number of Entity instances of the same type holding for condition
	Parent	Get	OBJECT	Returns the Parent Entity instance of this Entity instance. An Entity instance only has a Parent when it was created by a Split Activity. The original Entity instance entering the Split Activity is the Parent. Returns UNDEFINED if there is no Parent.
	Priority	Both	INTEGER	Priority of this Entity instance.
	Processing	Get	BOOLEAN	TRUE if the Entity is processing at an Activity.
	ReleaseTime	Get	REAL	Time Entity is released from an Activity or Process.
	SequenceNum	Get	INTEGER	A unique number given to each Entity as it is created.
	Type	Get	OBJECT	Entity Type reference of this Entity instance.
	WaitingForRe-source	Get	BOOLEAN	TRUE if the Entity is waiting to obtain one or more Resources.

SIMPROCESS System Attributes

Model Elements	Attribute Name[a]	Get/ Set	Attribute Type	Description
Resource	Name	Get	STRING	Name of the Resource.
	Activity	Get	OBJECT	Reference of Activity currently trying to assign or free this Resource (if any).
	Capacity	Get	REAL	Resource capacity.
	Entity	Get	OBJECT	Reference of current Entity instance being processed by the Activity trying to assign this Resource (if any).
	LastDowntime	Get	REAL	Duration of most recently initiated downtime.
	NumberWaiting	Get	INTEGER	Number of Entities waiting for this Resource.
	StartDowntime	Get	REAL	Time most recently initiated down-time started.
	UnitsBusy	Get	REAL	Number of units of the Resource in busy state.
	UnitsDown	Get	REAL	Number of units of the Resource not available because of the application of a downtime.
	UnitsIdle	Get	REAL	Number of units of the Resource in idle state.
Connector	Name	Get	STRING	Name of the Connector.
	NumberAccepted	Get	INTEGER	Number of Entities so far entered the Connector.
	NumberReleased	Get	INTEGER	Number of Entities exited the Con-nector.
	NumberIn	Get	INTEGER	Number of Entities currently on the Connector.
	LastDelay	Get	REAL	The value most recently initiated delay.

SIMPROCESS System Attributes

Model Elements	Attribute Name[a]	Get/ Set	Attribute Type	Description
	NextDelay	Set	REAL	Change the value of the next delay ONLY.
	AllFutureDelays	Set	REAL	Change the value of all future delays henceforth.
Activity's Group I	Name	Get	STRING	Name of the Activity.
	Parent	Get	OBJECT	Reference of this Activity's parent process.
	Entity	Get	OBJECT	Reference of the current Entity.
	Sibling(name: STRING)	Get	OBJECT	It returns the reference to the Activity or process with the specified name in the same hierarchical level. Returns UNDEFINED if not found.
Activity's Group II	NumberAccepted	Get	INTEGER	Number of Entities so far entered this Activity.
	NumberReleased	Get	INTEGER	Number of Entities exited this Activity.
	NumberIn	Get	INTEGER	Number of Entities currently at this Activity.
Activity's Group III	LastDelay	Get	REAL	The value of the most recently initiated delay.
	NextDelay	Set	REAL	Change the value of the next delay ONLY.
	AllFutureDelays	Set	REAL	Change the value of all future delays henceforth.
Assemble	Group I, II, & III			
	ActivatedByNoMatch	Get	BOOLEAN	Checks if the expression is activated by an Entity exiting through the NoMatch Pad.

SIMPROCESS System Attributes

Model Elements	Attribute Name[a]	Get/ Set	Attribute Type	Description
	ActivatedByTrig-ger	Get	BOOLEAN	Checks if the expression is activated by an Entity entering from the Trigger Pad.
	AssembleEntity-Type	Both	OBJECT	Type of Entity to create when an Assemble occurs.
	BatchEntities	Both	BOOLEAN	Value for whether to batch compo-nent Entities with assembled Entity.
	NumberWaiting	Get	INTEGER	Number of Entities at this activity waiting for resources.
	NumberOnHold	Get	INTEGER	Number of Entities at this activity holding for condition.
Assign	Group I, II, & III			
	NumberWaiting	Get	INTEGER	Number of Entities at this activity waiting for resources
Batch	Group I, II, & III			
	BatchEntityType	Both	OBJECT	Type of Entity to create when a Batch occurs. Note that when BatchEntity-Type is used to set the Entity Type for the Batch, the name of the Entity Type (STRING) instead of the Entity Type itself should be used. For exam-ple, BatchEntityType := “Entity1”.
	MaxBatchSize	Both	INTEGER	Number of Entities to batch.
	MinBatchSize	Both	INTEGER	Number of Entities must be in a batch before it can be released.
	MaxWaitTime	Both	REAL	Time to wait before releasing under-size batch.
	NumberWaiting	Get	INTEGER	Number of Entities at this activity waiting for resources
	NumberOnHold	Get	INTEGER	Number of Entities at this activity holding for condition

SIMPROCESS System Attributes

Model Elements	Attribute Name[a]	Get/ Set	Attribute Type	Description
Branch	Group I, II, & III			
	BranchName	Set	STRING	Name of the Connector to be taken by the next Entity.
	NumberWaiting	Get	INTEGER	Number of Entities at this activity waiting for resources
Clone	Group I, II, & III			
	NumOutEntities	Both	INTEGER	Number of Entities to output per Connector.
	NumberWaiting	Get	INTEGER	Number of Entities at this activity waiting for resources
Delay	Group I, II, & III			
	NumberWaiting	Get	INTEGER	Number of Entities at this activity waiting for resources
Dispose	Group I & II			
	MaxCount	Both	INTEGER	Number of Entities disposed to signal End Simulation event.
	NumberDisposed	Get	INTEGER	Number of Entities so far disposed at this Activity.
Free Resource	Group I & II			
Gate	Group I, II, & III			
	ActivatedByTrigger	Get	BOOLEAN	Checks if the expression is activated by an Entity entering from the Trigger Pad.
	AutoRelCount	Both	INTEGER	Number of Entities for threshold release.
	AutoRelease	Both	BOOLEAN	Whether threshold release is on.
	BatchEntities	Both	BOOLEAN	Value for whether to batch released Entities

SIMPROCESS System Attributes

Model Elements	Attribute Name[a]	Get/ Set	Attribute Type	Description
	BatchEntityType	Both	OBJECT	Type of Entity to create when a Batch occurs. Note that when BatchEntity-Type is used to set the Entity Type for the Batch, the name of the Entity Type (STRING) instead of the Entity Type itself should be used. For example, BatchEntityType := “Entity1”.
	LastRelCount	Get	INTEGER	Number of Entities released in the previous threshold/trigger release.
	TrigNextRel-Count	Both	INTEGER	Value of the trigger release quantity at the next trigger.
	TrigFutureRel-Count	Set	INTEGER	Change the trigger release quantity for all future triggered release henceforth.
	NumberWaiting	Get	INTEGER	Number of Entities at this activity waiting for resources
	NumberOnHold	Get	INTEGER	Number of Entities at this activity holding for condition
Generate	Group I			
	NumberGener-ated	Get	INTEGER	Number of Entities generated at this Activity.
Get Resource	Group I & II			
	NumberWaiting	Get	INTEGER	Number of Entities at this activity waiting for resources
Join	Group I, II, & III			
	BatchEntities	Both	BOOLEAN	Value for whether to batch family members with original Entity
	NumberWaiting	Get	INTEGER	Number of Entities at this activity waiting for resources
	NumberOnHold	Get	INTEGER	Number of Entities at this activity holding for condition

SIMPROCESS System Attributes

Model Elements	Attribute Name[a]	Get/ Set	Attribute Type	Description
Merge	Group I & II			
Process	Group I & II			
	Child(name : STRING)	Get	OBJECT	For hierarchical processes ONLY. It returns the reference to the child Activity or process with the specified name. Returns UNDEFINED if not found.
Replenish Resource	Group I & II			
Split	Group I, II, & III			
	NumberWaiting	Get	INTEGER	Number of Entities at this activity waiting for resources
Synchronize	Group I, II, & III			
	NumberWaiting	Get	INTEGER	Number of Entities at this activity waiting for resources
	NumberOnHold	Get	INTEGER	Number of Entities at this activity holding for condition
Transfer	Group I & II			
Transform	Group I, II, & III			
	NumOutEntities	Both	INTEGER	Number of output Entities.
	NumberWaiting	Get	INTEGER	Number of Entities at this activity waiting for resources
	TransformEntity-Type	Both	OBJECT	Type of Entity to create when a Transform occurs. Note that when TransformEntityType is used to set the Entity Type for the Transform, the name of the Entity Type (STRING) instead of the Entity Type itself should be used. For example, Trans-formEntityType := “Entity1”.

SIMPROCESS System Attributes

Model Elements	Attribute Name[a]	Get/ Set	Attribute Type	Description
Unbatch	Group I, II, & III			
	NumberWaiting	Get	INTEGER	Number of Entities at this activity waiting for resources
	RetainBatchedEntities	Both	BOOLEAN	Value of Retain Batched Entities
	UnbatchNestedBatches	Both	BOOLEAN	Value of Unbatch Nested Batches

a. System attributes' names must be input as shown. They are case sensitive.
b. Attributes in this category can be referenced in any expression.

SIMPROCESS System Methods

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
ABS	X: INTEGER or REAL	INTEGER or REAL	Returns the absolute value of X. The type (INTEGER or REAL) of the returned value will be the same type as the argument.
ActivateGenerate	Generate Name: STRING	NONE	If not already started, starts the generation of Entities by the specified Generate Activity.
ACOS	X: REAL	REAL	Returns the ACOSine of X.
AddPlotLegend	Plot: OJBECT Dataset: INTEGER Label: STRING Color (Optional): STRING	NONE	Adds a legend to a plot created through expressions (CreatePlot system method). Datasets are numbered beginning with 0. The Label of the legend must be specified. Specifying the Color is optional.
Alert	MasterEditor[b], Message: STRING	NONE	Display a message in the **Alert** dialog and wait for the user to close this dialog.
ASIN	X: REAL	REAL	Returns the ASine of X.
ATAN	X: REAL	REAL	Returns the ATANgent of X.
ATAN2	Y: REAL, X: REAL	REAL	Returns the ATANgent of X and Y.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
Attribute	AttributeName: STRING	OBJECT	DEPRECATED. Use GetAttributeValue and SetAttributeValue instead. Returns the reference to the specified Attribute. AttributeName follows the same convention as specifying Attributes in Expressions. That is, qualifiers (Entity, Entity.Type, and Model) might be required before the Attribute name (such as Entity.StartTime). The object owning the Attribute can be specified by including a colon (:) and the name of the object in AttributeName. For example Attribute("Time:Dispose1") would return the Attribute named Time from the Activity Dispose1. The object reference can be an Entity Type, Resource, or Activity. Note that using the object name is the only way to retrieve Resource Attributes.
BasicDateTime	None	STRING	Returns the current date and time of the simulation in MM/DD/YYYY HH:MM:SS format.
Beep	MasterEditor[b]	NONE	Makes a beep sound.
BINARYTOINT	BinaryValue: STRING	INTEGER	Returns the integer value of a binary string.
BREAKPOINT	None	NONE	Suspends simulation and displays simulation status. See "Method BREAKPOINT".
CEIL	X: REAL	INTEGER	Returns the smallest integer not less than X.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
ChangeColor	ActName: STRING, Color: STRING	NONE	Changes the color of the specified Activity. Entity references are no longer allowed. The keywords Self, Activity, and Parent or the name of an activity is allowed for ActName. Note that a keyword must be input as a STRING (such as "Self"). The Color can be any of the 65 colors listed in the SIMPROCESS Color Table at the end of this appendix. To clear a color setting enter "Clear" for the Color.
ClearMap	Map: OBJECT	NONE	Clears all values from the specified map.
ClearPlot	Plot: OBJECT	NONE	Clears the specified plot.
CloseDatabase	Database Connection: OBJECT	NONE	Closes the specified database connection.
CloseFile	File Stream: OBJECT	NONE	Closes the specified file.
CloseSpreadsheet	File Stream: OBJECT	NONE	Closes the specified spreadsheet file.
Confirm	MasterEditor[b], Message: STRING	BOOLEAN	Displays a message in the **Confirm** dialog and waits for the user to respond; **OK** returns TRUE, **CANCEL** returns FALSE.
ConvertToDateTime	NumberOfDays : REAL, INTEGER, or STRING	STRING	Converts a number representing the number of days since January 1, 1900, to a date and time in MM/DD/YYYY HH:MM:SS format.
Connector	ConnectorName: STRING	OBJECT	Returns the reference to the specified Connector. Returns UNDEFINED if not found.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
COS	X: REAL	REAL	Returns the COSine of X.
CreateActivityMap	None	OBJECT	Returns a map of all the Processes and Activities. The map key is the Name of each Process/ Activity so names must be unique to ensure the map contains all Processes and Activities in the model.
CreateArray	Type: STRING Dimension1: INTEGER Dimension2: INTEGER DimensionN: INTEGER	OBJECT	Returns an array of the specified type (“REAL”, “INTEGER”, “BOOLEAN”, “STRING”, or “ANYOBJ”) with the specified dimensions. For example, CreateArray(“REAL”, 2, 2), returns a 2 x 2 array for REAL numbers. Arrays of type REAL and INTEGER are initialized to zero. Arrays of type BOOLEAN are initialized to FALSE. Arrays of type STRING or ANYOBJ are not initialized, thus, attempting to retrieve a value from an array element that has no value will result in an error.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
CreateAttribute	Name: STRING; Type: STRING; Owner: STRING; Mode: STRING; DefaultValue: INTEGER, REAL, BOOLEAN, STRING, or UNDEFINED ArraySize (Optional): INTEGER DoNotReset (Optional): BOOLEAN StatType (Optional): STRING	NONE	Creates an Attribute. This method can only be used in the **`Start Run`** Expression (**Define/Model Expressions** or **Define Model Expressions** button on the Model Toolbar). Name should be a unique name that contains no spaces or special characters. An error will occur if an Attribute with the same Name and same Type already exists. Allowable Type values are Entity, Entity.Type, Resource, Activity, or Model. Owner should either be Global or the name of the Attribute owner. The Attribute owner can be an Entity Type, Resource, or Activity. Mode can be INTEGER, REAL, BOOLEAN, STRING, or OBJECT. The DefaultValue should match the designated Mode. If OBJECT is the Mode, UNDEFINED or "Null" should be entered for the DefaultValue. The remaining parameters are optional. They can be included in any order and not all have to be used. ArraySize sets the array dimension of the Attribute. A value of 0 indicates that the Attribute will not be an array Attribute. If DoNotReset is TRUE, the Attribute will not be reset between replications. This parameter only applies to global Attributes. It will be ignored for local Attributes. StatType can be Observation, TimeWeighted, or None. This parameter is ignored for Entity Attributes.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
CreateEntityType	Name: STRING Icon: STRING Priority: INTEGER Preempt: BOOLEAN Entity Stats: BOOLEAN	NONE	Creates an Entity Type. This method can only be used in the **`Start Run`** Expression (**Define/ Model Expressions** or **Define Model Expressions** button on the Model Toolbar). An error will occur if an Entity Type with the same name already exists in the model. “Default” can be used for the Icon parameter. Priority must be between 1 and 100 inclusive. Preempt sets the “Premept Lower Priority Entities” property, and Entity Stats sets the “Collect Entity Statistics” property.
CreateMap	Type: STRING	OBJECT	Returns a map of the specified type ("REAL", "INTEGER", "BOOLEAN", "STRING", or "ANYOBJ"). Maps are not initialized to default values.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
CreatePlot	Type: STRING Title: STRING X Axis Label (Optional): STRING Y Axis Label (Optional): STRING Bin Width (Optional): INTEGER, REAL	OBJECT	Creates plot and returns the Plot object to an Attribute of type Object. The Type can be Trace or Histogram. The Type and the Title are required. The axis labels are optional, but to have a Y Axis Label, there must be an X Axis Label. Bin Width is also optional and only applies to Histogram plots. It can be specified even if values for X Axis Label and Y Axis Label are omitted. AddPlotLegend is used to add legends to the Plot. PlotValue plots a new point. DisplayPlot causes the plot to be visible, and ClearPlot clears the plot of all data.
CreateResource	Name: STRING Units: REAL Fractional: BOOLEAN Consumable: BOOLEAN Resource Stats: BOOLEAN Resource by Activity Stats: BOOLEAN Resource by Shift Stats: BOOLEAN	NONE	Creates a Resource. This method can only be used in the **`Start Run`** Expression (**Define/Model Expressions** or **Define Model Expressions** button on the Model Toolbar). An error will occur if a Resource with the same name already exists in the model.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
CreateResourceShift	Name: STRING ResourceName: STRING UnitsAvailable: REAL DowntimePolicy: STRING ReleaseAll: BOOLEAN Weekday1: STRING StartTime1: REAL EndTime1: REAL Optionally to Weekday7: STRING StartTime7: REAL EndTime7: REAL	NONE	Creates a Resource Shift for a specific Resource. This method can only be used in the **`Start Run`** Expression (**Define/Model Expressions** or **Define Model Expressions** button on the Model Toolbar). The DowntimePolicy can be "Idle", "Interrupt", or "Empty". These options correspond to "Start Downtime when Resource is Idle", "Interrupt Activities", and "Start Downtime when Resource Queue Empty". "Interrupt Activities" includes the "Release All Resources" option. The ReleaseAll parameter sets this option. ReleaseAll is ignored for downtime policy options other than "Interrupt". There must be at least one set of Weekday, StartTime, and EndTime parameters. Allowable options for Weekday are "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", or "Saturday". StartTime and EndTime are in Hours. StartTime must be greater than or equal to 0.0, and EndTime must be less than or equal to 24.0. EndTime can be 0.0 only if StartTime is 0.0. If both are 0.0, then the complete 24 hour day is assumed. Since StartTime and EndTime must be REAL, 7:30 AM would be entered as 7.5. Up to seven groups of Weekday, StartTime, and EndTime can be included. Note that the groupings do not have to be in Weekday order.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
CreateWeeklyResourceDowntime	Name: STRING ResourceName: STRING UnitsDown: REAL Planned: BOOLEAN DowntimePolicy: STRING ReleaseAll: BOOLEAN Weekday1: STRING StartTime1: REAL EndTime1: REAL Optionally to Weekday7: STRING StartTime7: REAL EndTime7: REAL	NONE	Creates a Weekly Resource Downtime for a specific Resource. This method can only be used in the **`Start Run`** Expression (**Define/Model Expressions** or **Define Model Expressions** button on the Model Toolbar). The DowntimePolicy can be “Idle”, “Interrupt”, or “Empty”. These options correspond to “Start Downtime when Resource is Idle”, “Interrupt Activities”, and “Start Downtime when Resource Queue Empty”. “Interrupt Activities” includes the “Release All Resources” option. The ReleaseAll parameter sets this option. ReleaseAll is ignored for downtime policy options other than “Interrupt”. There must be at least one set of Weekday, StartTime, and EndTime parameters. Allowable options for Weekday are “Sunday”, “Monday”, “Tuesday”, “Wednesday”, “Thursday”, “Friday”, or “Saturday”. StartTime and EndTime are in Hours. StartTime must be greater than or equal to 0.0, and EndTime must be less than or equal to 24.0. EndTime can be 0.0 only if StartTime is 0.0. If both are 0.0, then the complete 24 hour day is assumed. Since StartTime and EndTime must be REAL, 7:30 AM would be entered as 7.5. Up to seven groups of Weekday, StartTime, and EndTime can be included. Note that the groupings do not have to be in Weekday order.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
DateTime	None	STRING	Returns the current date and time of the simulation in MM/DD/YYYY HH:MM:SS:MSEC:USEC:NSEC format.
DayOfWeek	None	STRING	Returns the current day of the week (Monday, Tuesday, etc.).
DeactivateGenerate	Generate Name: STRING	NONE	Ends the generation of Entities by the specified Generate Activity.
DecreaseCapacity	Resource Name: STRING Units: INTEGER or REAL	NONE	Decreases the capacity of a Resource by the units specified.
DeleteFile	File Name: STRING	BOOLEAN	Delete the specified file or directory. Returns TRUE if the action is successful. The File Name indicates where the file or directory is located. If it does not contain any path separator characters (i.e., backslashes on Windows, slashes on other systems), it is assumed to be in a folder named for the model in the same location from which the model was opened. If path separators are used, the program assumes the user is providing a valid absolute path. Note that Windows paths require double backslashes instead of single backslashes.
DisplayPlot	Plot: OBJECT	NONE	Displays the specified plot. A Plot can also be displayed by using the Menu (**Report/Display Real-Time Plots**) or by using the **Display Plots** button on the tool bar.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
DisposeResultSet	ResultSetName: STRING	NONE	Disposes a Result Set created by ReadFromDatabase.
DrawIntegerSample	Statistical Distribution: STRING	INTEGER	Draws a sample of type integer from the specified statistical distribution.
DrawRealSample	Statistical Distribution: STRING	REAL	Draws a sample of type real from the specified statistical distribution.
ElapsedTime	Units (Optional): STRING Start: STRING End: STRING	REAL	Returns the amount of time between the Start value and End value. The Units is optional and can be any valid time unit ("Seconds," "Minutes," etc.). If omitted, the value returned is in the units set by **Simulation Time Unit** in **Run Settings**. Start and End use the format MM/DD/YYYY HH:MM:SS:MSEC:USEC:NSEC. The formats MM/DD/YY and YYYY-MM-DD can also be used for the date portion of Start and End. The **DateTime** function can be used for Start or End.
EntityExists	Gate: OBJECT SequenceNum: INTEGER	INTEGER	Checks the specified Gate activity (obtained using the Gate System Method) for an Entity with the specified SequenceNum. Returns the position number of the Entity in the Gate. This can be used with the GetEntity System Method to return a reference to the Entity. If the Entity is not being held in the Gate activity, 0 is returned.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
EXP	X: REAL	REAL	Returns e raised to the power of X.
ExternalCall	Class : STRING or OBJECT Method Name: STRING Method Parameters (as required by external method): INTEGER, REAL, STRING, BOOLEAN, OBJECT	BOOLEAN, REAL, INTEGER, STRING, or OBJECT	Executes a method of an external Java class. Assumes the class is in the appropriate package structure in a **`classes`** directory within the model's directory or in a properly constructed jar file in the same location. (See "SIMPROCESS and External Java Classes" for other options.) Errors in the calls will result in an aborted simulation. Note: To use this call you must have purchased the External Application Interface plug-in license from CACI.
FireTrigger	Activity Name: STRING	NONE	Fires the trigger of the specified Gate or Assemble Activity.
FLOAT	Number: INTEGER	REAL	Converts the argument to REAL.
FLOOR	X: REAL	INTEGER	Returns the largest integer not greater than X.
FreeResource	Resource Name: STRING Tag (Optional): STRING Consume (Optional): BOOLEAN	NONE	Creates a free Resource request for an Activity.
Function	Function Name: STRING Arg1Value (Optional) Arg2Value (Optional) . . . ArgNValue (Optional): REAL, INTEGER, BOOLEAN, STRING	INTEGER, REAL, BOOLEAN, STRING	Returns the value of a User-defined Function. Optional argument values can be included. Local variables of the form Arg1, Arg2,..., ArgN must be defined in the Function. See "User-Defined Functions" for more information.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
Gate	Gate Name: STRING	OBJECT	Returns a reference to the specified Gate activity. Returns UNDEFINED if not found.
GenerateEntity	Generate Name: STRING Entity Name (Optional): STRING Quantity (Optional): INTEGER Time (Optional): REAL	NONE	Generates Entities from the specified Generate Activity. If Entity Name is not specified, the default defined for Entity Type in the specified Generate Activity is used. If Quantity is not specified or the Quantity is zero or less, the default Quantity in the designated Generate Activity is used. If Time is not specified, the action occurs at the current simulation time. Time is in the **Simulation Time Units** set in the **Run Settings**. Each of the optional parameters can be specified without including any of the other optional parameters.
GetActivityStatistic	Activity Name: STRING Statistic: STRING Value Type: STRING Replication: INTEGER	REAL	Returns the value of the requested **Activity Name** statistic. **Statistic** is one of the statistics types for Activities listed in the Labels column of the table in Appendix H. (See "Statistic Types".) **Value Type** is either "Avg," "StDev," "Min," "Max," "Count," or "RunLength." A **Replication** of -1 returns the average of replications, and a **Replication** of -2 returns the sum of replications.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
GetActivityByEntityStatistic	Activity Name: STRING Entity Name: STRING Statistic: STRING Value Type: STRING Replication: INTEGER	REAL	Returns the value of the requested **Activity Name** and **Entity Name** statistic. **Statistic** is one of the statistics types for Activities listed in the Labels column of the table in Appendix H. (See "Statistic Types".) **Value Type** is either "Avg," "StDev," "Min," "Max," "Count," or "RunLength." The value for **Replication** cannot be larger than the current replication simulating. A **Replication** of -1 returns the average of replications, and a **Replication** of -2 returns the sum of replications.
GetArrayValue	Array: OBJECT Index1: INTEGER Index2: INTEGER IndexN: INTEGER	REAL, INTEGER, BOOLEAN, STRING, or OBJECT	Returns from the designated array the value at the specified indices. Note that the array indices are zero based. Attempting to get a value from an element of a STRING or ANYOBJ array that has had no value assigned will result in an error.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
GetAttributeStatistic	Attribute Name: STRING Value Type: STRING Replication: INTEGER	REAL	Returns the value of the requested **Attribute Name**. **Attribute Name** must be unique among the attributes that are being statistically monitored. **Value Type** is either "Avg," "StDev," "Min," "Max," "Count," or "Run-Length." A **Replication** of -1 returns the average of replications, and a **Replication** of -2 returns the sum of replications.
GetAttributeValue	AttributeName: STRING	REAL, INTEGER, BOOLEAN, STRING, or OBJECT	Returns the value of the specified Attribute. AttributeName follows the same convention as specifying Attributes in Expressions. That is, qualifiers (Entity, Entity.Type, and Model) might be required before the Attribute name (such as Entity.StartTime). The object owning the Attribute can be specified by including a colon (:) and the name of the object in AttributeName. For example Attribute("Time:Dispose1") would return the Attribute named Time from the Activity Dispose1. The object reference can be an Entity Type, Resource, or Activity. Note that using the object name is the only way to retrieve Resource Attributes.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
GetConnectorStatistic	Connector Name: STRING Statistic: STRING Value Type: STRING Replication: INTEGER	REAL	Returns the value of the requested **Connector Name** statistic. **Statistic** is one of the statistics types for Connectors listed in the Labels column of the table in Appendix H. (See "Statistic Types".) **Value Type** is either "Avg," "StDev," "Min," "Max," "Count," or "RunLength." A **Replication** of -1 returns the average of replications, and a **Replication** of -2 returns the sum of replications.
GetConnectorByEntityStatistic	Connector Name: STRING Entity Name: STRING Statistic: STRING Value Type: STRING Replication: INTEGER	REAL	Returns the value of the requested **Connector Name** and **Entity Name** statistic. **Statistic** is one of the statistics types for Connectors listed in the Labels column of the table in Appendix H. (See "Statistic Types".) **Value Type** is either "Avg," "StDev," "Min," "Max," "Count," or "RunLength." The value for **Replication** cannot be larger than the current replication simulating. A **Replication** of -1 returns the average of replications, and a **Replication** of -2 returns the sum of replications.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
GetCostStatistic	Period Name: STRING Cost Object 1: STRING Cost Object 2: STRING Cost Object 3 (Optional): STRING; Cost Type: STRING Value Type: STRING Replication: INTEGER	REAL	Returns the value of the requested **Period Name**, **Cost Object 1**, **Cost Object 2**, and optionally **Cost Object 3**. **Cost Object 1** can be the name of a Resource, an Activity, or an Org-Model. **Cost Object 2** can be the name of an Activity, an Entity, or an Org Node. **Cost Object 3** is only used when requesting an Org Node by Resource cost. Thus, the combination of **Cost Object 1** and **Cost Object 2** can be "Resource and Entity," "Resource and Activity," "Activity and Entity," or "OrgModel and Org Node." If **Cost Object 1** is an OrgModel name, **Cost Object 2** must be an Org Node name and **Cost Object 3** must be a Resource name. **Cost Type** options are "Variable" or "Absorption". **Value Type** is either "Avg," "StDev," "Min," or "Max" for **Replication** -1. If **Replication** is 1 or greater, the maximum value is returned no matter what is entered for **Value Type**. A **Replication** of -1 returns the average of replications, and a **Replication** of -2 returns the sum of replications.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
GetEntity	Gate, Resource, or Entity: OBJECT Index: INTEGER	OBJECT	Returns a reference to the Entity being held at the specified Gate Activity, waiting for the specified Resource, or being held as part of the specified Entity's batch. Index indicates the position in the appropriate queue and must be greater than or equal to 1 and less than or equal to the number of Entities being held at the Gate activity (obtained by using Gate System Attribute NumberOnHold), less than or equal to the number of Entities waiting for the Resource (obtained by using Resource System Attribute NumberWaiting), or less than or equal to the number of Entities in the batch (obtained by using the Batch System Attribute BatchSize).
GetEntityBySequenceNum	SequenceNum: INTEGER	OBJECT	Returns a reference to the Entity specified by SequenceNum. Returns UNDEFINED if the specified Entity does not exist.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
GetEntityStatistic	Entity Name: STRING Statistic: STRING Value Type: STRING Replication: INTEGER	REAL	Returns the value of the requested **Entity Name** statistic. **Statistic** is one of the statistics types for Entities listed in the Labels column of the table in Appendix H. (See "Statistic Types".) **Value Type** is either "Avg," "StDev," "Min," "Max," "Count," or "RunLength." A **Replication** of -1 returns the average of replications, and a **Replication** of -2 returns the sum of replications.
GetFromMap	Map: ANYOBJ Key: INTEGER, REAL, BOOLEAN, STRING, ANYOBJ	INTEGER, REAL, BOOLEAN, STRING, ANYOBJ	Returns from the designated map the value referenced by the specified key.
GetInputSourceValue	Input Source: STRING	REAL, BOOLEAN, STRING	Returns the next value from a File, Spreadsheet, or Database Input Source. If the value is a number a REAL number will be returned even if the number has no fractional portion. A BOOLEAN will be returned if the value from the File, Spreadsheet, or Database equals (ignoring case) TRUE or FALSE. All other values are returned as a STRING. See "Input Sources" for more information on Input Sources.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
GetMapKey	Map: ANYOBJ Index: INTEGER;	INTEGER, REAL, BOOLEAN, STRING, ANYOBJ	Returns from the designated map the key specified by the index.
GetMapSize	Map: ANYOBJ	INTEGER	Returns the number of objects in the specified map.
GetNext	Result Set Name: STRING	BOOLEAN	Used to advance to the next row of a Result Set. Returns TRUE if the Result Set has more values.
GetResource	Resource Name: STRING Units: STRING, INTEGER, or REAL Tag (Optional): STRING	NONE	Creates a get Resource request for an Activity.
GetResourceStatistic	Resource Name: STRING Statistic: STRING Value Type: STRING Replication: INTEGER	REAL	Returns the value of the requested **Resource Name** statistic. **Statistic** is one of the statistics types for Resources listed in the Labels column of the table in Appendix H. (See "Statistic Types".) **Value Type** is either "Avg," "StDev," "Min," "Max," "Count," or "RunLength." A **Replication** of -1 returns the average of replications, and a **Replication** of -2 returns the sum of replications.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
GetResourceByActivityStatistic	Resource Name: STRING ActivityName: STRING Value Type: STRING Replication: INTEGER	REAL	Returns the value of the requested **Resource Name** and **Activity Name** statistic. **Value Type** is either "Avg," "StDev," "Min," "Max," "Count," or "RunLength." A **Replication** of -1 returns the average of replications, and a **Replication** of -2 returns the sum of replications.
GetResourceByShiftStatistic	Resource Name: STRING Shift Name: STRING Value Type: STRING Replication: INTEGER	REAL	Returns the value of the requested **Resource Name** and **Shift Name** statistic. **Value Type** is either "Avg," "StDev," "Min," "Max," "Count," or "RunLength." A **Replication** of -1 returns the average of replications, and a **Replication** of -2 returns the sum of replications.
GetResult	Result Set Name: STRING Database Field: STRING	INTEGER, REAL, BOOLEAN, STRING	Returns a value from a Result Set. Type of value returned is dependent on the type of values in the requested field of the Result Set.
GetTimeStampStatistic	Start Key: STRING StopKey: STRING Value Type: STRING Replication: INTEGER	REAL	Returns the value of the Time Stamp with **Start Key** and **Stop Key**. **Value Type** is either "Avg," "StDev," "Min," "Max," "Count," or "RunLength." A **Replication** of -1 returns the average of replications, and a Replication of -2 returns the sum of replications.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
GetTotalCostStatistic	Name: STRING Cost Type: STRING Value Type: STRING Replication: INTEGER	REAL	Returns the total cost for the requested Resource, Entity, or Activity **Name**. If **Name** is "Total," then the total cost of all Resources is returned. **Cost Type** is either "Variable" or "Absorption." **Value Type** is either "Avg," "StDev," "Min," or "Max" for **Replication** -1. If **Replication** is 1 or greater, the maximum value is returned no matter what is entered for **Value Type**. A **Replication** of -1 returns the average of replications, and a **Replication** of -2 returns the sum of replications.
HALT	Message (Optional): STRING	NONE	If used with no parameters, it ends the current replication and simulation continues with next replication. If parameters are used, the simulation is stopped with an error condition. A dialog appears that contains the contents of the parameter.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
INCLUDE	File Name: STRING	NONE	Specifies the name of an external file that contains Expression statements. If no path is included then the file is assumed to be in the model's directory. The Expression statements contained in the file replace the INCLUDE statement. The File Name parameter can be either a STRING or a Model Attribute of type STRING. When a Model Attribute is used, the file name returned from the Model Attribute is the default value assigned when the Model Attribute was created. If a different value is desired, the Model Attribute must be designated as a Model Parameter and the value changed at simulation start.
IncreaseCapacity	Resource Name: STRING Units: INTEGER or REAL	NONE	Increases the capacity of a Resource by the units specified.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
INPUT	Message: STRING Variable: INTEGER, REAL, BOOLEAN, STRING Location (Optional): STRING	NONE	Reads a single value into Variable from a dialog prompt. Variable can be a local variable, user-defined Attribute, or System Attribute that is type Set or Both. The name of Variable will be in the dialog title, and the Message will be displayed. If Message is an empty String (""), a default Message will be displayed. The value entered will be validated. (For example, an Integer value must be entered for a Variable with type INTEGER.) Location to display the dialog is optional. The Location must be one of the major points on the compass (N, S, E, W, NE, NW, SE, SW) or center (C). If no Location is specified, the dialog will be centered.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
InterruptActivity	Activity Name: STRING Receive Transfer: STRING Entity State (Optional): STRING Entity State (Optional): STRING	INTEGER	Interrupts the Activity (or Activities if multiple Activities have the same name) specified by Activity Name. If no Entity State parameters are specified or an Entity State parameter is "All, Entities processing, waiting for Resources, or holding for a condition are interrupted. Entities that are already interrupted due to Resource downtime or Entity preemption are not affected. The Receive Transform parameter is the name of the Local Receive Transform Activity that will receive the interrupted Entities. Up to two Entity State parameters may be specified. Allowable values are "All", "Processing", "WaitingForResource", or "HoldingForCondition". If the specified Activity is a Process, all Activities within the Process are interrupted. Returns the number of Entities interrupted. (See "Method InterruptActivity".)
InterruptEntity	SequenceNum: INTEGER Receive Transfer: STRING	BOOLEAN	Interrupts the Entity specified by SequenceNum. The Receive Transform parameter is the name of the Local Receive Transform Activity that will receive the interrupted Entity. Returns TRUE if the interrupt is successful. (See "Method InterruptEntity".)
INTTOBINARY	number: INTEGER	STRING	Converts a non-negative integer number to a binary string.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
INTTOSTR	number: INTEGER	STRING	Converts an integer number to a string.
InvokeWebService	ContinueOnError: BOOLEAN or STRING WSDL URL: STRING Service Name: STRING Service Port Type Name: STRING Operation Name: STRING Parameter Values (Optional): STRING, BOOLEAN, INTEGER, REAL, or ANYOBJ	STRING, BOOLEAN, INTEGER, REAL, or ANYOBJ	Invokes an operation (or method) on a Web Service. See "Method InvokeWebService" for more information
LN	X: REAL	REAL	Returns the Natural Log of X.
LOG10	X: REAL	REAL	Returns the base 10 Log of X.
MapKeyExists	Map: ANYOBJ Key: INTEGER, REAL, BOOLEAN, STRING, ANYOBJ	BOOLEAN	Returns TRUE if the designated Map contains the specified Key.
MAXOF	list of numbers: REAL/INTEGER (all the numbers in list must be of the same type)	REAL or INTEGER	Returns the maximum number in the given list. Return type is the same as the input list.
MINOF	list of numbers: REAL/INTEGER (all the numbers in list must be of the same type)	REAL or INTEGER	Returns the minimum number in the given list. Return type is the same as the input list.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
OpenApplication	Application: STRING File Name (Optional): STRING	NONE	Opens the specified Application and, optionally, the specified File Name within the Application. The Application parameter must include the full path to the application's executable file. A full path is not required if an application recognized by SIMPROCESS is used. For example, if Excel is the default spreadsheet application for opening the Standard Report in a spreadsheet, "Excel" can be used for Application. The optional File Name indicates where the file to open is located. If it does not contain any path separator characters (i.e., backslashes on Windows, slashes on other systems), it is assumed to be in a folder named for the model in the same location from which the model was opened. If path separators are used, the program assumes the user is providing a valid absolute path. Note that Windows paths require double backslashes instead of single backslashes.
OpenDatabase	Properties File: STRING StopOnError (Optional): BOOLEAN	OBJECT	Creates a connection to the database specified in the Properties File. The Properties File is the name of the file that specifies the URL and protocol to use. When StopOnError is used and is FALSE, if an error occurs the System Attribute UNDEFINED is returned, and the simulation is not stopped.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
OpenFile	Mode: STRING File Name: STRING StopOnError: (Optional): BOOLEAN	OBJECT	Open the specified file and return the file stream object. Mode can be "input" or "output." The File Name String can contain path names if the user wants to explicitly tell SIMPROCESS where the file resides. If path separators (e.g., \ in Windows, / on other systems) are used, the program assumes the user is providing a complete and valid file path. If no path separators are detected in the file name, the Model's folder is where the files will reside. Note that double backslashes (\\) must be used in Windows file paths instead of a single backslash. When StopOnError is used and is FALSE, if an error occurs the System Attribute UNDEFINED is returned, and the simulation is not stopped.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
OpenSpreadsheet	Mode: STRING File Name: STRING StopOnError (Optional): BOOLEAN	OBJECT	Open the specified spreadsheet file and return the file stream object. Mode can be "input" or "output." The File Name String can contain path names if the user wants to explicitly tell SIMPROCESS where the file resides. If path separators (e.g., / on other systems or \ in Windows) are used, the program assumes the user is providing a complete and valid file path. If no path separators are detected in the file name, the Model's folder is where the files will reside. Note that double backslashes (\\) must be used in file paths instead of a single backslash. When StopOnError is used and is FALSE, if an error occurs the System Attribute UNDEFINED is returned, and the simulation is not stopped.
OUTPUT	Takes zero or more arguments separated by commas. Type can be REAL, BOOLEAN, INTEGER, or STRING	NONE	Display the specified arguments in the **Output** dialog. Functions (including nested functions) that result in any of the argument types can be used in the OUTPUT statement.
PlotValue	Plot: OBJECT Dataset: INTEGER X Value: INTEGER or REAL Y Value (Trace only): INTEGER or REAL	NONE	Adds a data point to the specified Dataset of the plot. Datasets are numbered beginning with 0. Trace plots require an X Value and a Y Value. Histogram plots only use the X Value.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
POSITION	String1: STRING String2: STRING	INTEGER	This method returns the starting position of String2 in String1. If String2 is not completely contained in String1, it returns 0.
POWER	X: REAL, Y REAL	REAL	Returns X raised to the power of Y.
PutInMap	Map: ANYOBJ Key: INTEGER, REAL, BOOLEAN, STRING, ANYOBJ Value: INTEGER, REAL, BOOLEAN, STRING, ANYOBJ	NONE	Places the value in the specified map using the key as reference.
READ	INTEGER, REAL, STRING, or BOOLEAN	INTEGER, REAL, STRING, or BOOLEAN.	Reads a single variable into an identifier from a dialog prompt. The name of the identifier will be presented in the dialog for the user, and the value entered will be validated (i.e., INTEGER type must be an integer value).
ReadFromDatabase	Database Connection: OBJECT Result Set Name: STRING SQL Statement: STRING	NONE	Reads values based on the SQL Statement from the database specified by the connection. The values are placed in a Result Set with the given Result Set Name. The Result Set Name should be unique for each ReadFromDatabase unless a Result Set is to be replaced.
ReadFromFile	File Stream: OBJECT List of variables	NONE	Read values for the listed variables from the input file associated with the specified File Stream.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
ReadFromSpread-sheet	File Stream: OBJECT Sheet Name: STRING Cell Reference: STRING Variable: STRING, INTE-GER, REAL, or BOOLEAN	NONE	Read a value into the variable represented by the Value parame-ter from the input spreadsheet file associated with the specified File Stream. Cell Reference is a single String that contains the typical spreadsheet method for identify-ing a cell (e.g. “A1”).
ReadFromSpread-sheet	File Stream: OBJECT Sheet Name: STRING Row: INTEGER Column: INTEGER Variable: STRING, INTE-GER, REAL, or BOOLEAN	NONE	DEPRECATED. Use ReadFrom-Spreadsheet with Cell Reference instead. Read a value from the specified sheet, row, and column of the input spreadsheet file asso-ciated with the specified File Stream. The value is assigned to Variable.
REALTOSTR	number: REAL precision: INTEGER (Option-al)	STRING	Converts a real number to a string. If a precision is specified, the number is rounded to the specified precision. Trailing ze-roes are added if the decimal part of the number is less than the specified precision.
ReleaseEntity	Gate: OBJECT SequenceNum: INTEGER	BOOL-EAN	Causes the release of the Entity with the specified SequenceNum from the specified Gate. (The Gate parameter is obtained using the Gate System Method.) Re-turns TRUE if the Entity was re-leased, FALSE otherwise.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
RemoteCall	Server URL: STRING Server Name: STRING Method Name: STRING Method Parameters (as required by remote method): INTEGER, REAL, STRING, BOOLEAN	BOOLEAN	Executes a method of a remote application. Assumes a Java RMI server has been registered with the objects and methods being called. Errors in the calls will result in an aborted simulation. Note: To use this call you must have purchased the External Application Interface plug-in license from CACI.
RemoveFromMap	Map: ANYOBJ Key: INTEGER, REAL, BOOLEAN, STRING, ANYOBJ	INTEGER, REAL, BOOLEAN, STRING, ANYOBJ	Removes and returns from the designated map the value referenced by the specified key.
REPLACE	String1: STRING position1, position2: INTEGER String2: STRING	STRING	Returns a string that is in String1 with part of String1 from position 1 to position 2 replaced with String2.
Resource	ResourceName: STRING	OBJECT	Returns the reference to the specified Resource.
ROUND	number: REAL	INTEGER	Rounds the argument and returns the closest integer value.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
SetActivityExpression	ActivityName: STRING ExpressionType: STRING Expression: STRING	NONE	Applies SIMPROCESS Expression code in the specified Expression parameter to the specified Activity. This method can only be used in the **`Start Run`** Expression (**Define/Model Expressions** or **Define Model Expressions** button on the Model Toolbar). ExpressionType is case sensitive must be valid for the type of Activity specified. The Expression parameter should be the complete Expression code in a single STRING or the complete file name with no path of the file that contains the Expression code. If the Expression parameter is a file, the file must located in the model's directory.
SetArrayValue	Array: OBJECT Index1: INTEGER Index2: INTEGER IndexN: INTEGER Value: REAL, INTEGER, BOOLEAN, STRING, or OBJECT	NONE	Sets the value at the specified indices of the specified array. Note that the indices are zero based.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
SetAttributeValue	AttributeName: STRING Value: REAL, INTEGER, BOOLEAN, STRING, or OBJECT	NONE	Sets the value of the specified Attribute. AttributeName follows the same convention as specifying Attributes in Expressions. That is, qualifiers (Entity, Entity.Type, and Model) might be required before the Attribute name (such as Entity.StartTime). The object owning the Attribute can be specified by including a colon (:) and the name of the object in AttributeName. For example Attribute("Time:Dispose1") would return the Attribute named Time from the Activity Dispose1. The object reference can be an Entity Type, Resource, or Activity. Note that using the object name is the only way to retrieve Resource Attributes.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
SetEntityExpression	EntityTypeName: STRING ExpressionType: STRING Expression: STRING	NONE	Applies SIMPROCESS Expression code in the specified Expression parameter to the specified Entity Type. This method can only be used in the **`Start Run`** Expression (**Define/Model Expressions** or **Define Model Expressions** button on the Model Toolbar). ExpressionType is case sensitive and must be a valid Entity Instance Expression type (**`Initialize Entity`**, **`Accept Entity`**, **`Interrupt Processing`**, **`Resume Processing`**, **`Release Entity`**, or **`Dispose of Entity`**). The Expression parameter should be the complete Expression code in a single STRING or the complete file name with no path of the file that contains the Expression code. If the Expression parameter is a file, the file must located in the model's directory.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
SetEntityTypeExpression	EntityTypeName: STRING ExpressionType: STRING Expression: STRING	NONE	Applies SIMPROCESS Expression code in the specified Expression parameter to the specified Entity Type. This method can only be used in the **`Start Run`** Expression (**Define/Model Expressions** or **Define Model Expressions** button on the Model Toolbar). ExpressionType is case sensitive and must be a valid Entity Type Expression type (**`Start Simulation`**, **`Start Trial`**, **`Initialize Entity`**, **`Dispose of Entity`**, **`End Trial`**, or **`End Simulation`**). The Expression parameter should be the complete Expression code in a single STRING or the complete file name with no path of the file that contains the Expression code. If the Expression parameter is a file, the file must located in the model's directory.
SetRandomNumberOption	Option: STRING	NONE	Sets the type of random numbers to use for the simulation. This method can only be used in the **`Start Run`** Expression. Allowable options are "Legacy", "Mersenne Twister", or "MT" (same as Mersenne Twister).

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
SetReplications	NumReps: INTEGER	NONE	Sets the number of replications for the model. Overrides the **Number of Replications** value in the **Run Settings**. This method can only be used in the **`Start Run`** Expression (**Define/Model Expressions** or **Define Model Expressions** button on the Model Toolbar).
SetResourceCost	ResourceName: STRING Cost: REAL CostType: STRING TimeUnit: STRING	NONE	Sets one of the costs of the specified Resource. CostType can be PerEntity, PerUnit, PerTimeUnit, or Fixed. If PerTimeUnit is the CostType, then TimeUnit must be one of the valid SIMPROCESS time units (Hours, Minutes, etc.). If Fixed is the CostType, then the TimeUnit must be Weekly, Montly, Quarterly, Half-Yearly, or Yearly. TimeUnit is not required if the CostType is PerEntity or PerUnit. Note that if the Resource is consumable, then only PerEntity and PerUnit are valid CostTypes.
SetResourceDowntime	ResourceName: STRING DowntimeName: STRING	NONE	Applies a global Resource Downtime (**Define/Resource Downtimes**) to the specified Resource. This method can only be used in the **`Start Run`** Expression (**Define/Model Expressions** or **Define Model Expressions** button on the Model Toolbar).

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
SetResourceExpression	ResourceName: STRING ExpressionType: STRING Expression: STRING	NONE	Applies SIMPROCESS Expression code in the specified Expression parameter to the specified Resource. This method can only be used in the **`Start Run`** Expression (**Define/Model Expressions** or **Define Model Expressions** button on the Model Toolbar). ExpressionType is case sensitive and must be a valid Resource Expression type (**`Start Simulation`**, **`Start Trial`**, **`Get Resource`**, **`Free Resource`**, **`Start Downtime`**, **`End Downtime`**, **`End Trial`**, or **`End Simulation`**). The Expression parameter should be the complete Expression code in a single STRING or the complete file name with no path of the file that contains the Expression code. If the Expression parameter is a file, the file must located in the model's directory.
SetResourceShift	ResourceName: STRING ShiftName: STRING	NONE	Applies a Resource Shift (**Define/Resource Shifts**) to the specified Resource. This method can only be used in the **`Start Run`** Expression (**Define/Model Expressions** or **Define Model Expressions** button on the Model Toolbar).

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
SetSimulationStart-End	Start: STRING End: STRING	NONE	Sets the Start Date/Time and/or the End Date/Time for the model. Overrides the Start Date/Time and End Date/Time values in the **Run Settings**. This method can only be used in the Start Run Expression (**Define/Model Expressions** or **Define Model Expressions** button on the Model Toolbar). An empty STRING ("") for either argument indicates that the Date/Time from the **Run Settings** should be used for that argument. Start and End use the format MM/DD/YYYY HH:MM:SS:MSEC:USEC:NSEC. The formats MM/DD/YY and YYYY-MM-DD can also be used for the date portion of Start and End. The Time portion of Start or End can be omitted. If omitted, midnight is assumed. Also, if Time is included, values for smaller time units can be omitted. For example, the time portion can just be 08:00. All time values to nanoseconds are not required.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
SetStartEnd	Name: STRING Start: STRING End: STRING	NONE	Sets the Start Date/Time and/or the End Date/Time of a Generate Activity, a specific Entity generation schedule of a Generate Activity, the Downtimes of a Resource, or a global Resource Downtime. Overrides the Start Date/Time and End Date/Time values specified in those places. This method can only be used in the Start Run Expression. The Name parameter must be the name of a Generate Activity, a Resource (modifies all the Downtime schedules assigned to the Resource), or a global Resource Downtime (modifies the Downtime of all Resources assigned to this global Resource Downtime). To specify the name of a Generate Schedule use the name of the Generate Activity combined with the name of the Generate Schedule using a colon in between (for example "Generate1:Periodic1"). An empty STRING ("") for Start or End indicates that the Date/Time from the Run Settings should be used for that argument. Start and End use the format MM/DD/YYYY HH:MM:SS:MSEC:USEC:NSEC. The formats MM/DD/YY and YYYY-MM-DD can also be used for the date portion of Start and End. The Time portion of Start or End can be omitted. If omitted, midnight is assumed. Also, if Time is included, values for smaller time units can be omitted. For example, the time portion can just be 08:00. All time values to nanoseconds are not required.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
SetWarmupLength	Length: REAL Units (Optional): STRING	NONE	Sets the warmup length for the model. Overrides the **Warmup Length** value in the **Run Settings**. This method can only be used in the **`Start Run`** Expression (**Define/Model Expressions** or **Define Model Expressions** button on the Model Toolbar). The Units argument is optional and can be any valid time unit (Seconds, Minutes, etc.). If omitted, the time unit value of **Warmup Time Unit** in **Run Settings** is assumed.
ShowSystemAttributes[c]	None	NONE	Display all the System-Defined Attributes of the calling element and any related elements that are within scope (i.e., if this function is called on the Activity expression, then the system attributes for any Entity or Resource elements will also be included) in the **Output** dialog.
ShowUserAttributes[c]	None	NONE	Display all the User-Defined Attributes evaluated for the calling elements and any related elements that are within scope (i.e., if this function is called on the Activity expression, then the user-defined attributes for any Entity or Resource elements will also be included) in the **Output** dialog.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
SimTime	Timeunit (Optional): STRING Valid time units are - Years, Months, Weeks, Days, Hours, Minutes, Seconds, Milliseconds, Microseconds, Nanoseconds. Time unit strings are case sensitive.	REAL	Return current simulation time in time units specified. If no parameter is provided, time unit defaults to the Simulation Time Unit set in the Run Settings.
SIN	X: REAL	REAL	Returns the Sine of X.
SOAPCall	ContinueOnError: BOOLEAN or STRING Endpoint URL: STRING Service Name: STRING Service Port Type Name: STRING Body namespace URI: STRING Method namespace URI: STRING SOAP action URI: STRING Method to invoke: STRING Parameters (Optional): Parameter Name : STRING Parameter Value: STRING, BOOLEAN, INTEGER, REAL	STRING	Invokes methods (or operations) on Web Services. See "Method SOAPCall" for more information.
SQRT	X: REAL	REAL	Returns the square root of X.
STATUS	Same as OUTPUT	NONE	Display the specified arguments in status bar.
StopSimulation	NONE	NONE	Completely stops a running simulation. Any remaining replications are not simulated.
STRLEN	inString: STRING	INTEGER	Returns the length of the string, inString.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
STRTOINT	inString: STRING	INTEGER	Converts a string consisting only of numbers with no decimal in the string to an integer.
STRTOREAL	inString: STRING	REAL	Converts a string of numbers including a decimal to a real.
SUBSTR	position1, position2: INTEGER String: STRING	STRING	Returns the substring of String from position1 to position2, inclusive. Note that the first position is 1 instead of 0.
SystemTime	None	REAL	Returns the current system time in milliseconds as the difference between the current time and midnight, 1 Jan 1970 UTC.
TAN	X: REAL	REAL	Returns the TANgent of X.
TimeOfDay	Timeunit (Optional): STRING Valid time units are - Years, Months, Weeks, Days, Hours, Minutes, Seconds, Milliseconds, Microseconds, Nanoseconds. Time unit strings are case sensitive.	REAL	Returns the current time of day in the requested time unit. If no parameter is provided, time unit defaults to the Simulation Time Unit set in the Run Settings.
TRUNC	number: REAL	INTEGER	Returns the integer part of the number
UpdateBackgroundGraphic	Background Graphic Name: STRING Background Image Name: STRING	NONE	Update a Background Graphic to show a named background image.
UpdateDashboardLabel	Dashboard Name: STRING Host or IP Address: STRING Port: INTEGER TextLabel Name: STRING Value: INTEGER, REAL, STRING or BOOLEAN	NONE	Update the specified TextLabel with the given value.

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
UpdateDynamicLabel	MasterEditor[b] Label Name: STRING Label ID: INTEGER Font Color: STRING Value: Same as Label's Mode	NONE	Update the specified dynamic label with the given value.
WriteToDatabase	Database Connection: OBJECT SQL Statement: STRING Check Warnings (Optional): BOOLEAN	NONE	Used to modify tables of the database specified by the connection. The SQL Statement should be of type INSERT, DELETE, or UPDATE. After the execution of the the SQL Statement, the default is to check for SQL warnings. Check Warnings should be used and set to FALSE to suppress the check for SQL warnings.
WriteToDatabase	Name: STRING Replication: INTEGER	NONE	Executes a predefined SQL statement with the requested replication. (See "Building SQL Statements" for information on defining a SQL statement.) Replication must be between 1 and the number of replications defined for the simulation or be -1 for average of replication results.
WriteToFile	File Stream: OBJECT list of variables	NONE	Write the values of the listed variables to the output file specified by the File Stream.
WriteToSpreadsheet	File Stream: OBJECT Sheet Name: STRING Cell Reference: STRING Variable: STRING, REAL, INTEGER, or BOOLEAN	NONE	Write the value of the Value parameter to the output spreadsheet file specified by the File Stream. Cell Reference is a single String that contains the typical spreadsheet method for identifying a cell (e.g. "A1").

SIMPROCESS System Methods

Method Name[a]	Arguments	Return	Description
WriteToSpreadsheet	File Stream: OBJECT Sheet Name: STRING Row: INTEGER Column: INTEGER Variable: STRING, REAL, INTEGER, or BOOLEAN	NONE	DEPRECATED. Use WriteToSpreadsheet with Cell Reference instead. Writes the value of Variable to the specified sheet, row, and column of the output spreadsheet file specified by the File Stream.
WriteToSpreadsheet	Name: STRING Replication: INTEGER	NONE	Executes a predefined spreadsheet with the requested replication. (See "Building a Spreadsheet Export" for information on defining a spreadsheet.) Replication must be between 1 and the number of replications defined for the simulation or be -1 for average of replication results.

a. System methods' names must be input as shown. They are case sensitive.
b. This argument (i.e., MasterEditor) must be the first argument inside the parentheses of this method. It is required by the method and must be input as shown. It is case sensitive.
c. Since this method does not require any argument, do not include empty parentheses when using this method.

System Method Examples

Many of the System Methods described here are demonstrated with models in the **ExpressionDemos** directory. Included in that directory is *SIMPROCESS Expression Demo Models*, which describes each model. If this directory is not part of the **SIMPROCESS/models** directory, it can be downloaded at www.simprocess.com.

Methods Alert and Beep

For example, to display a message and sound alarm when an Entity enters an Activity, put the following lines in the Activity's **Accept Entity** event:

```
Beep(MasterEditor);
Alert(MasterEditor, "Press OK to continue!");
```

Methods Confirm, DrawRealSample, MINOF, MAXOF, and REALTOSTR

This example demonstrates the syntax and usage of the listed methods. It draws a number from a Uniform distribution whose minimum and maximum parameters are results from an Exponential distribution with a mean of 30. The result is displayed in a message box with **OK** and **Cancel** buttons. Pressing **OK** will continue this process again, while **Cancel** will end the process.

```
fContinue : BOOLEAN;
value1, value2, maxval, minval : REAL;

fContinue := TRUE;
WHILE fContinue
value1 := DrawRealSample("Exp(30.0)");
value2 := DrawRealSample("Exp(30.0)");
maxval := MAXOF(value1, value2);
minval := MINOF(value1, value2);
value1 := DrawRealSample("Uni("+REALTOSTR(minval)+"," +REALTO-
   STR(maxval)+")");
Beep(MasterEditor);
fContinue := Confirm(MasterEditor, "The value is "+ REALTO-
   STR(value1));
END WHILE;
```

Method ChangeColor

This method changes the color of a non-bitmap icon. Currently, Activity icon "RectangleIcon," and Process icon "DefaultIcon" can change color. The "SIMPROCESS Color Table" contains the 65 colors available in SIMPROCESS. The name of an Activity or Process can be specified, or "Self", "Parent", or "Activity" can be used to designate the Activity or Process. Note that in earlier versions of SIMPROCESS (pre 4.1), "Self", "Parent", and "Activity" did not require quotes. Earlier version models using those keywords must be modified to include quotes for use with SIMPROCESS 4.1 or later versions. If an Activity or Process name is used, SIMPROCESS will attempt to change the color of every Activity or Process that matches the input name. To clear a color setting, enter "Clear" for the color.

The following changes the color of the Activity Delay4 to Red.

```
ChangeColor("Delay4", "Red");
```

The following changes the color of the current Activity or Process to Blue.

```
ChangeColor("Self", "Blue");
```

The following changes the color of the parent Process of the current Activity or Process to Yellow.

```
ChangeColor("Parent", "Yellow");
```

The following clears the color setting of the Activity.

```
ChangeColor("Sales Process", "Clear");
```

The ability to change colors of Entities is no longer supported. For example, the following code will now produce an error.

```
ChangeColor(Entity, "Brown");
```

To change the color of the Activity icon to green if there is no Entity in it, in the Activity's **`Release Entity`** event add:

```
IF (NumberIn = 0)
   ChangeColor("Self", "Green");
END IF;
```

Methods ShowSystemAttributes and ShowUserAttributes

These methods display all the system-defined and user-defined attributes with their current values of the calling element in the standard output window. For example, to display the system-defined attributes of an Activity when an Entity enters, in the Activity's **Accept Entity** event put:

```
ShowSystemAttributes;
```

All the system-defined attributes and values are listed in the standard output window. In addition, any related elements' system attributes or user-defined attributes will be listed in the output window as well. For example, if the ShowUserAttributes method is called from the expression script in an Activity event (such as Accept Entity), the Entity that is within scope will also have its user-defined attributes listed. The same goes for Resources being assigned.

Model level user-defined attributes are listed no matter what elements are within scope.

Method SimTime

This method returns the current simulation time in hours unless a time unit parameter is provided. The return value is **REAL** type. For example:

```
Entity.StartTime := SimTime; { StartTime is a REAL global
   Entity instance attribute of type REAL. }
```

The allowable time units are provided as a string parameter and are case sensitive. The allowable time units are: Years, Months, Weeks, Days, Hours, Minutes, Seconds, Milliseconds, Microseconds, Nanoseconds. An example using the time units is as follows:

```
Mytimeunit : STRING;
Mytimeunit:="Weeks";
OUTPUT("My time unit in weeks = ",
    SimTime(Mytimeunit);
OUTPUT("My time unit in months = ",
    SimTime("Months");
```

Methods OUTPUT and STATUS

To display a message in the standard output window:

```
OUTPUT("Number of orders currently held in Gate10 is : ", Num-
   berIn);

realnum : REAL;

OUTPUT("The String value of realnum = ",

   REALTOSTR(realnum));
```

Or, to display it in the status bar:

```
STATUS("Number of orders currently held in Gate10 is : ", Num-
   berIn);
```

where the expressions above are placed on the Activity, Gate10.

Method UpdateDashboardLabel

To use this method, you need to assign a Dashboard to your model which contains a TextLabel. To assign Dashboards, refer to "Assigning Dashboards".

Five arguments are required for this method. The first is **`Dashboard Name`**. This is the name used when you assigned the Dashboard to your model. You may also use the name of the Dashboard's file (without any path and without the ".spd" extension), although this is not recommended since it negatively impacts performance. The second argument is the **`Host or IP Address`** where a Dashboard Server is expected to be receiving messages for the assigned Dashboard. This value must match the one entered when the Dashboard was assigned. It is important to note that SIMPROCESS compares the values and not their meanings. While **`localhost`** and **`127.0.0.1`** refer to the same system (i.e., they have the same meaning), they are not the same values to SIMPROCESS. The third argument is the **`port`** on which a Dashboard Server is expected to be receiving messages for the assigned Dashboard. This value must also match the one entered when the Dashboard was assigned. The fourth argument is the name of a Graph on the assigned Dashboard of type TextLabel. The name is an uneditable field displayed on the properties dialog for the TextLabel when assigning a Dashboard and will be unique for that Dashboard. The fifth argument is the value to be displayed on the TextLabel. It can be of type **`INTEGER`**, **`REAL`**, **`STRING`** or **`BOOLEAN`**.

For example, the `Call Center` model has assigned the `Call Status.spd` Dashboard file using the name `Call Status`. The assigned Dashboard designated a Server Address of `localhost` and port `5555`. This Dashboard contains a TextLabel by the name of `TextLabel4`. During simulation, you want to display the number of Entities generated at a Generate Activity. In the Release Entity expression of the selected Generate Activity add the following line:

```
UpdateDashboardLabel("Call Status", "localhost", 5555, "TextLa-
   bel4", NumberGenerated);
```

While this statement may be used with any TextLabel on any assigned Dashboard, it is best used with those that have no Plot Values assigned to it. In that way, you may be certain that the value you place on the TextLabel via this statement will remain unchanged until a subsequent statement updates it.

Method UpdateDynamicLabel

To use this method, you need to define a dynamic label in the work area first. To define a dynamic label, refer to "For a list of System Methods available in SIMPROCESS, refer to "SIMPROCESS System Methods"."

Five arguments are required for this method. The first one is **`MasterEditor`**. This is the key word that must be there and input as shown. The next two arguments are the **`Name`** and **`ID`** of the dynamic label defined in the **Background Text Properties** dialog. The fourth argument is the color to be used in displaying this label. The last argument is the value to be displayed. It can be a constant or an attribute, but the data type must match the **Mode** defined in **Background Text Properties**.

For example, you want to display the number of Entities generated at a Generate Activity. Assume you have already defined a dynamic label with the name "total," ID of 0, and data type of INTEGER. In the Release Entity event of the Generate Activity add the following line:

```
UpdateDynamicLabel(MasterEditor, "total", 0, "DarkBlue", Num-
   berGenerated);
```

The first parameter of this method must be **`MasterEditor`** and input as shown.

Methods OpenFile, CloseFile, ReadFromFile, and WriteToFile

`OpenFile` opens the file specified in the **Filename** field for either input or output. The mode must be either "input" or "output" (not case-sensitive). "Input" opens a file to read, while "output" opens a file to write. Any other specifications will lead to an error message. Both arguments are STRING type, i.e., they must be surrounded by quotation marks. **`OpenFile`** returns an Object type value which is required in **`CloseFile, ReadFromFile`**, and **`WriteToFile`** methods.

```
OpenFile(Mode, Filename) : OBJECT;
```

By default, SIMPROCESS will look for either input or output files in the directory of the active model. Either file can be placed elsewhere as long as the path is specified in the **`OpenFile`** statement. Note that double backslashes (\\) must be used instead of single backslashes (\) in Windows file paths.

The following example opens a file (`myoutput.dat`) for output. First, define a Model Attribute **`MyOutStream`** that is type Object. If the file is to be opened in the **`Start Run`** Expression, make sure that **Do Not Reset Before Each Replication** is selected on the attribute properties dialog. If the file is opened in the Start Simulation or Start Trial expression, the file will be opened each replication.

Type the following statement in one of the expressions.

```
Model.MyOutStream := OpenFile("Output", "myoutput.dat");
```

The next example opens a file (`myinput.dat`) for input and keeps the return value in the model attribute **`MyInStream`**.

```
Model.MyInStream := OpenFile("input", "myinput.dat");
```

`ReadFromFile` reads the specified numbers of values from a file that is open for input. The first argument is the input stream, from a previous **`OpenFile`** call (the file must be open for input). From the previous example, the model attribute, **`Model.MyInStream`** would be the input stream. Following the stream argument is a list of variables whose values are going to be read in from the file.

```
ReadFromFile(Input Stream, List of
   variables);
```

In the input file, values should be separated by one or more white spaces (i.e., space, tab, or new line characters). There is also a pre-defined special delimiter "|" in SIMPROCESS. You can use "|" to separate values. This delimiter is very useful to separate values (strings) that contain any white space. You can surround the whole string with a pair of "|"s, and **`ReadFromFile`** will read in whatever is in between as one value.

The input file can contain comments. Any characters that follow double slashes (**`//`**) will be ignored.

The type of value to be read in is dependent on the specified variable type. A mismatch will cause an error. The following table shows the valid types for different value types. Integer and real are any valid numerical values. **`ReadFromFile`** will either truncate a real value to integer for an Integer type variable, or float an integer value to real for a Real type variable. String is a sequence of characters. If spaces are included, surround the whole string with "|". Boolean can only be either TRUE or FALSE. Any other value will cause an error. Boolean values are not case sensitive.

	Value Types in File			
Variable Types	Integer	Real	String	Boolean
Integer	OK	Truncated	Error	Error
Real	FLOATed	OK	Error	Error
String	OK	OK	OK	OK
Boolean	Error	Error	Error	OK
Object	Error	Error	Error	Error

Example:

The input file contains:

```
|This is a test for ReadFromFile.|

123      67.89    TRUE
```

The expression reads as:

```
StrVal    : STRING;
IntVal    : INTEGER;
RealVal   : REAL;
BoolVal : BOOLEAN;
ReadFromFile(Model.MyInStream, StrVal, IntVal);
ReadFromFile(Model.MyInStream, RealVal, BoolVal);
```

The expression will read in the following:

```
StrVal has the value "This is a test for ReadFromFile.".
```

IntVal has the value **123**.

RealVal has the value **67.89**.

BoolVal has the value **TRUE**.

WriteToFile writes the values of the specified variables to a file that is open for output. The first argument is the output stream, that is the returned value from a previously **OpenFile** call (must be open for output). Following the stream argument is a list of variables whose values are going to be written to a file.

```
WriteToFile(Output Stream, List of variables);
```

The output values will be written to the file consecutively. Any kind of format or spacing has to be output explicitly by the user. There are two special characters that can be used in formatting outputs: "^" will write a tab to the file, while "/" will advance to a new line. For these to work correctly, they must be written individually and be surrounded by "".

For example:

Following the prior example of **ReadFromFile**, an expression has:

```
WriteToFile(Model.MyOutStream, StrVal, "/", IntVal, "^");
WriteToFile(Model.MyOutStream, RealVal, "^", BoolVal);
```

Write to the file as:

```
This is a test for ReadFromFile.

123      67.89    TRUE
```

CloseFile closes the file the IOStream pointed to.

CloseFile(IOStream);

For example:

```
CloseFile(Model.MyOutStream);
```

CloseFile(Model.MyInStream);

Close the files `myoutput.dat` and `myinput.dat,` respectively.

Typically, you will place **CloseFile** in the End Trial Expression, especially if you put **OpenFile** in the Start Trial Expression.

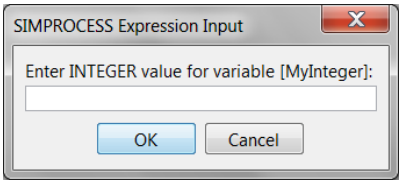
Method READ

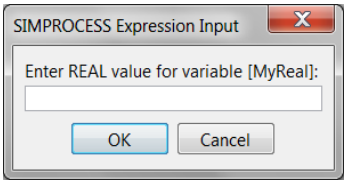
READ prompts the user for a single variable input. (Also see "Method INPUT".) The variable must be a locally-defined variable (i.e., a system attribute or a user-defined attribute such as a Model, Activity, Resource, or Entity attribute cannot be referenced directly). Example:

```
MyInteger : INTEGER;
MyREAL : REAL;
READ MyInteger;
READ MyREAL;
```

The above example will provide an input prompt requesting the user to input an Integer value for **MyInteger**, and the next read statement will prompt the user to input a REAL value for **MyREAL**. Type checking is done on the values entered so the user should make sure the values are compatible (i.e., don't enter "hello dolly" when prompted to enter an integer). If the value is not compatible, an error exception is thrown and the simulation stops.

The prompts for the above example would look as follows:



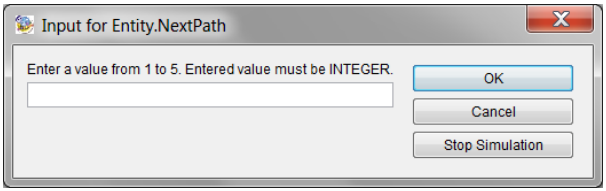


Method INPUT

INPUT prompts the user for a single variable input. The variable must be a locally-defined variable, user-defined Attribute, or System Attribute with type Set or Both. Unlike READ, INPUT allows the user to include their own message. For example, assume the Entity Attribute NextPath is an INTEGER Attribute.

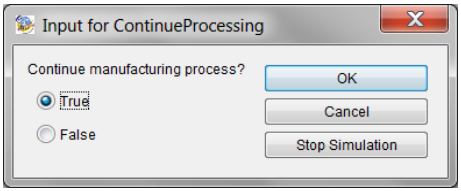
```
INPUT("Enter a value from 1 to 5." Entity.NextPath);
```

The above example will provide an input prompt requesting the user to input an Integer value for **Entity.NextPath**. Type checking is done on the value entered to ensure the values are compatible. (Only enter an Integer if the variable is type INTEGER.) If the value is not compatible, an error is displayed and a new value must be entered. In the example above the message says that the value should be between 1 and 5. The INPUT statement will not validate that the value is within that range. It will only validate that the entered value is an INTEGER. If an empty String ("") is used for the message, a default message will display. There is also an option to stop the simulation. Below is how the above example displays.



Note that appended to the message is a reminder to enter an INTEGER. A similar reminder is added

when the variable is type REAL. If the variable is type STRING, no validation occurs. If the variable is type BOOLEAN, only the selection of True or False is offered.



Method HALT

HALT terminates the current replication or simulation. When used with no parameters, the current replication is ended and the next replication begins. This is similar to using the **Maximum Entity Count** on the Dispose Activity. **HALT** with parameters terminates the simulation in an error condition. The value of the parameters are displayed in an error dialog.

This statement is useful for debugging and stopping simulations at certain conditions in the model when developing a complex model. You may prefer the use of **OUTPUT** and **ALERT** statements to notify you of happenings in the model, but the **HALT** is there if you need it. Example:

```
HALT;
```
(ends current replication)

```
HALT("Encountered error in Delay5");
```
(terminates simulation)

Method RemoteCall

RemoteCall is a plug-in capability that can be licensed separately from CACI to allow users to make calls to external applications from within the expression scripts of a SIMPROCESS model. This feature is a powerful way to make calls to user applications (either custom developed or commercial products) based on how the business process model and workflow is planned. An example of this might be a model that represents a "To Be" business process flow for Human Resources where you intend to use a commercially available human Resources product such as Peoplesoft, Lawson, or Oracle. Based on specific Activities in the model execution, you could make expression script calls to a user-written server that would "feed" the commercial application as a prototype to see if the product will align with the desired business process needs. The server program could "feed" the application either from native calls to the product or via other interface technologies such as XML. This capability will support analysis using models between the business functions and the system transactions that are required

to support them.

The **RemoteCall** provides the mechanisms to make calls to a Java RMI server that has been registered using the Java RMI Registry. It is the responsibility of the user to develop the RMI server program and start the application. The expression script method simply has to have the name of the server computer (could be the same computer that the model is running on or could be a separate server over the network), the server object to be called, the remote method to call, and the parameter list that matches the remote procedure in the user-written server.

Example:

```
mybool : BOOLEAN;
mybool:=RemoteCall("rmi://localhost/", "server-demo", "putPlot-
 Data", SimTime, Self.NumberAccepted);
```

In the example above, a remote server application (this one happens to be running on the same computer as the model, hence the localhost URL address) called server-demo is looked up using the Java RMI registry, and the putPlotData method is called with two parameters - SimTime and NumberAccepted, where NumberAccepted is the number of Entities accepted in the Activity. (See "System Attributes".)

The following is a description of the server application showing the actual Java code needed to implement this example. This text is not a complete reference to RMI programming, and we recommend you reference any of the many Java-published materials on RMI (java.sun.com is a good place to start).

The following is a list of the source Java programs that demonstrate the RemoteCall interface. The files are located in the **SPUser/SampleFiles** directory.

SPServerDemo.java - defines the server class and the methods that will be called from the expression script. The Java code is as follows

```
package com.caci.demo;

import java.rmi.*;

import java.util.*;

import ptolemy.plot.*;

import ptolemy.plot.plotml.*;


public interface SPServerDemo extends Remote

{  boolean putPlotData(double simtime, int totalEntities )
```

```
        throws RemoteException;
    boolean clearData()
        throws RemoteException;
}
```

SPServerDemoImpl.java - defines the implementation code for the server class and methods. This example instantiates a user-defined class that uses the Ptolemy plotting classes. This plot is being presented on the screen to the user by the server program separate from the SIMPROCESS simulation. The data being fed to the plot is coming from the simulation and the RemoteCall parameters shown above. The Java code is as follows:

```
package com.caci.demo;
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
import ptolemy.plot.*;
import ptolemy.plot.plotml.*;

public class SPServerDemoImpl
    extends UnicastRemoteObject
    implements SPServerDemo
{
public SPServerDemoImpl()
        throws RemoteException
    {
    myplot = new SPPlotDemo("Number Entities", "Simulation Time
in Hours");
    }
```

```
    public boolean putPlotData(double simtime, int totalEntities )
        throws RemoteException
      {
        myplot.putTicks(simtime, totalEntities);
        return true;
      }
      public boolean clearData()
        throws RemoteException
      {
        myplot.clearData();
        return true;
      }
      private String name;
      SPPlotDemo myplot;
  }
```

SPServerDemoServer.java - defines the server binding to the RMI registry. This is necessary for Java client code (such as the RemoteCall in SIMPROCESS) to do a lookup and find the server by a name that was "binded" to the actual server class that is running on the remote server. The Java code is as follows:

```
package com.caci.demo;
import java.rmi.*;
import java.rmi.server.*;
import ptolemy.plot.*;
import ptolemy.plot.plotml.*;
```

```
public class SPServerDemoServer
{  public static void main(String args[])
   {  try
      {  System.out.println
            ("Constructing Server implementations...");
         SPServerDemoImpl d1
            = new SPServerDemoImpl();
         System.out.println
           ("Binding server implementations to registry...");
         Naming.rebind("serverdemo", d1);
         System.out.println
            ("Waiting for invocations from clients...");
      }
      catch(Exception e)
      {  System.out.println("Error: " + e);
      }
   }
}
```

SPPlotDemo.java - defines the plotting class that is instantiated in the **SPServerDemoImpl.java** class. The Java code is as follows:

```
package com.caci.demo;
import ptolemy.plot.*;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
```

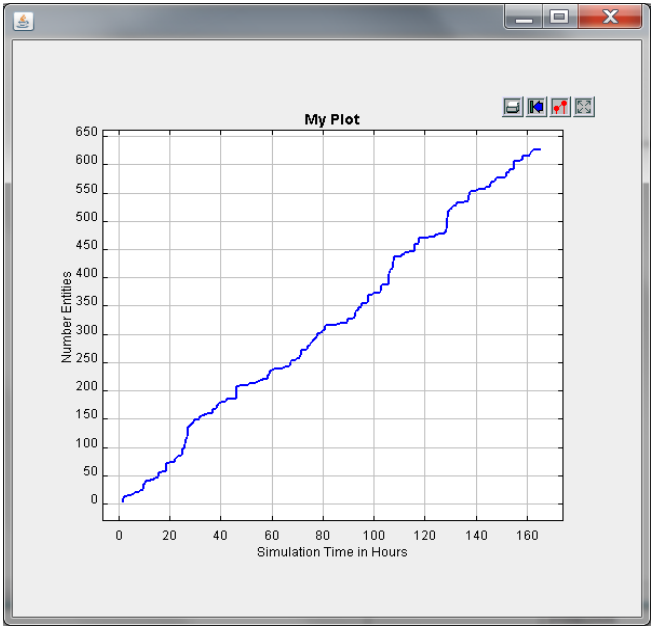
```
import javax.swing.JFrame;

public class SPPlotDemo extends JFrame {
Plot myPlot = null;
boolean first = true;

SPPlotDemo(String yLabel, String xLabel) {
        // Instantiate the plot.
        myPlot = new Plot();
        // Set the size of the toplevel window.
        this.setSize(600, 600);
         myPlot.setSize(550, 550);
        myPlot.setButtons(true);
        myPlot.setTitle("My Plot");
        myPlot.setXLabel(xLabel);
        myPlot.setYLabel(yLabel);
        myPlot.setYRange(0.0, 1.0);
        myPlot.setXRange(0.0, 20.0);
        myPlot.setMarksStyle("none");
        myPlot.setImpulses(false,1);
  myPlot.setConnected(true,1);
  myPlot.setBars(false);

  myPlot.setSize(350,300);
        GridBagLayout gridbag = new GridBagLayout();
        GridBagConstraints c = new GridBagConstraints();
```

```
            getContentPane().setLayout(gridbag);
            c.gridx = 0;
            c.gridy = 0;
            c.gridwidth = 1;
            gridbag.setConstraints(myPlot, c);
            getContentPane().add(myPlot);
            show();
        }
    public void putTicks(double time, int num)
    {
          myPlot.addPoint(1, time, num, !first);
          first = false;
          myPlot.fillPlot();
            }
    public void clearData()
    {
          myPlot.clear(true);
          myPlot.fillPlot();
            }
    }
```

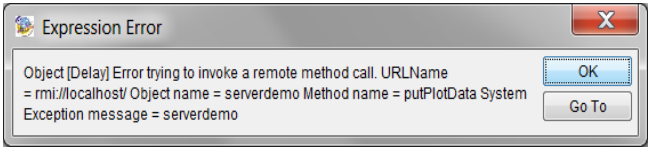
You should reference your RMI programming documentation for details on how to compile and start the server code as well as the RMI registry on a server computer. "Displaying Plots Remotely" has examples on starting the RMI registry.

Once you have started the RMI registry and the SPServerDemoServer program, you can run the demonstration model RemoteCall.spm to demonstrate this capability. (Chapter 5 of the *SIMPROCESS Getting Started Manual* (Demonstration and Reference Models) has instructions on running this example.) When you start the simulation, the server program will put up a plot window and plot the values (**`SimTime`** and **`Number Accepted`** in this example) on the plot window. The window will

appear as follows on the server:



If the server is not running, your simulation will be stopped and you will receive a message dialog stating that the server connection could not be made - similar to the following:


Expression Error
Object [Delay] Error trying to invoke a remote method call. URLName
= rmi://localhost/ Object name = serverdemo Method name = putPlotData System
Exception message = serverdemo
OK
Go To

If your parameters don't match in number and type, RMI Exceptions will occur and a message dialog will be presented with the error. You will notice in this example that the putPlotData method expects a Java double parameter and an int parameter. The following is a mapping of the SIMPROCESS data types that are allowed on the RemoteCall parameters and how they map to Java data types:

SIMPROCESS	Java
REAL	double
STRING	String

INTEGER	int
BOOLEAN	boolean

Method ExternalCall

`ExternalCall` is a plug-in capability that can be licensed separately from CACI to allow users to make calls to external Java classes from within the expression scripts of a SIMPROCESS model. When calling an external Java class from an Expression using this statement, it may be necessary to add compiled Java class files to the classpath used by SIMPROCESS. See "SIMPROCESS and External Java Classes" for information on how to add external Java classes to the SIMPROCESS classpath.

Since **`ExternalCall`** is a function, a value will be returned. If the method called in the external class does not return a value (return type is void), SIMPROCESS will return the value TRUE; thus, the method should be assigned to a BOOLEAN local variable or attribute. Only valid SIMPROCESS data types can be returned (STRING, BOOLEAN, INTEGER, REAL, ANYOBJ). (For types other than ANYOBJ, see previous table in RemoteCall section for comparable Java data types. ANYOBJ can be any valid Java class.)

The ExternalCall function requires two parameters: class or class name and method name. Any parameters after the first two are considered to be parameters to the method being called. The first parameter must be the fully qualified name of a class, such as "**`com.acme.stuff.MyClass`**," or it may be a previously created class instance stored in an Attribute of type Object. If a class name is provided and the second parameter is the simple name of the class, then a constructor with no arguments will be called and the result returned (any additional parameters will be ignored). It should then be stored in an Attribute of type Object for use in future calls, as SIMPROCESS will not retain a reference to the newly created instance. The following is an example of invoking a constructor:

```
Model.MyClass := ExternalCall("com.acme.stuff.MyClass",
"MyClass");
```

In subsequent calls, the instance can be used as in this example:

```
MyString : STRING;

MyString := ExternalCall(Model.MyClass, "getString");
```

Method names must be unique for the class. That is, SIMPROCESS looks for a method on the class by name only. The number of parameters is not considered. The named method must take the same number and type of arguments, in order, as those provided in additional parameters to ExternalCall. It's possible to invoke static methods on either an instance of the class or by providing its qualified name. (See **`ExternalCall.spm`** in the **`ExpressionDemos`** directory.)

Some examples:

```
MyString : STRING;

MyString := ExternalCall("com.acme.stuff.MyClass", "getString");
```

This example returns a string value. The "getString" method has no parameters.

```
number : REAL;

number := ExternalCall("com.acme.stuff.MyClass", "getSquare",
   100.0);
```

This example returns a REAL (double) value. The "getSquareRoot" method has one parameter (double).

```
boolval : BOOLEAN;

boolval := ExternalCall(Model.MyClass, "returnNothing");
```

Note that the method does not return a value (return type is void), so the function must be assigned to a BOOLEAN variable. Also, this example uses a previously instantiated class stored in the Model Attribute MyClass.

Method ActivateGenerate

ActivateGenerate starts the generation of Entities by a Generate Activity. For this method to work correctly, the Generate Activity must not be active. This is accomplished by setting the start date (**Start/End** tab) to a date that is past the end date of the simulation. For instance, if the simulation is scheduled to end on 1/1/2005, then the start date of the Generate Activity must be 1/1/2005 or later. Note that this is the start date of the Generate Activity, not the start date of a particular schedule within the Generate Activity. When the appropriate conditions are met in the simulation for the Generate Activity to start, issue the command along with the name of the Generate Activity. It is important that the name of the Generate Activity be unique.

```
ActivateGenerate("Generate2");
```

The **ExpressionDemos** directory has a sample model named **ActivateGenerate.spm**. If this directory is not part of the **SIMPROCESS/models** directory, it can be downloaded at www.simprocess.com.

Methods ReleaseEntity, Gate, GetEntity, EntityExists

These methods are used in conjunction with Gate Activities. **ReleaseEntity** is used to release a specific Entity that is being held at a Gate Activity. **Gate** returns a reference to a Gate Activity. **GetEntity** returns a reference to an Entity being held in a Gate Activity. (**GetEntity** can also be used to return a reference to an Entity waiting for a Resource or return a reference to an Entity held

in another Entity's batch.) **EntityExists** checks for the presence of a specific Entity at a Gate Activity. For example:

```
gate : ANYOBJ;
entity : ANYOBJ;
i : INTEGER;
goodRelease : BOOLEAN;
gate := Gate("Gate12"); {get reference to Gate activity}
FOR i := 1 TO gate.NumberOnHold
  entity := GetEntity(gate, i); {get the entity at position i}
  IF entity.Attribute2 > 100.0
    {release the entities that match the condition}
    goodRelease := ReleaseEntity(gate, entity.SequenceNum);
    IF NOT goodRelease
      {notify of entity not released}
      OUTPUT("Entity ", entity.SequenceNum, "not released from Gate
      ", gate.Name);
    END IF;
  END IF;
END FOR;
```

EntityExists can be used to find the index position of a specific entity. For example (using the same Gate reference as the above example).

```
position : INTEGER;
{42 is the SequenceNum of the Entity desired}
position := EntityExists(gate, 42);
IF position > 0
  entity := GetEntity(gate, position);
  goodRelease := ReleaseEntity(gate, 42);
END IF;
```

The **ExpressionDemos** directory has a sample model named **ReleaseEntity.spm**. If this directory is not part of the **SIMPROCESS/models** directory, it can be downloaded at www.simprocess.com.

Methods CreateArray, GetArrayValue, SetArrayValue

These methods are used to create and manipulate multi-dimensional arrays. **CreateArray** creates a REAL, INTEGER, BOOLEAN, STRING, or ANYOBJ array. Arrays of type REAL and INTEGER

are initialized to zero. BOOLEAN arrays are initialized to FALSE. Arrays of type STRING and ANYOBJ are not initialized with any values. **GetArrayValue** retrieves a value from an array, and **SetArrayValue** places a value in an array. (Note that if **GetArrayValue** is used with a STRING or ANYOBJ array in which the requested element has not had a value assigned by **SetArrayValue**, an error will occur.) An Attribute with a **Mode** of Object must be defined to create an array. For example, assume there is a Model Attribute named TimeArray with **Mode** Object.

```
Model.TimeArray := CreateArray("REAL", 3, 2);
```

This creates a two dimensional array for REAL numbers. The first dimension has a length of 3, and the second dimension has a length of 2. The type of array (REAL, INTEGER, etc.) must be entered as a STRING. This array can contain the delay times for 6 routes (3 origin locations and 2 destination locations). Use **SetArrayValue** to populate the array. Note that array indices are zero based. Also, the type of the value passed to the array must match the type of the array.

```
i : INTEGER;
j : INTEGER;
value : REAL;
FOR i := 0 TO 2
  FOR j := 0 TO 1
    value := DrawRealSample("Log(10, 1, 10)");
    SetArrayValue(Model.TimeArray, i, j, value);
  END FOR;
END FOR;
```

Values are retrieved from an array using **GetArrayValue**. Assume the three origin locations are represented by 0, 1, and 2, and the two destination locations are represented by 0 and 1. These are assigned to the Entity Attributes **Origin** and **Destination** (local variables would also work).

```
Entity.Origin := DrawIntegerSample("Int(0, 2, 11)");
Entity.Destination := DrawIntegerSample("Int(0, 1, 12)");
Entity.DelayTime := GetArrayValue(Model.TimeArray, Entity.Origin,
Entity.Destination);
```

Since arrays can also be of type ANYOBJ, SIMPROCESS object references (such as Activities, Entities, and Resources) can be stored in an array. For instance, a particular Activity stored in an array could be referenced in any Expression since a Model Attribute can be used in any Expression. Assume a **CreateArray** statement created a one dimensional ANYOBJ array assigned to the Model Attribute **ActivityArray**.

```
myActivity : ANYOBJ;
myActivity := Sibling("Delay12");
SetArrayValue(Model.ActivityArray, 0, myActivity);
```

The **ExpressionDemos** directory has a sample model named **CreateArray.spm**. This directory can be downloaded at www.simprocess.com.

Method SOAPCall

The SOAPCall statement provides SIMPROCESS models with the ability to invoke methods (or operations) on Web Services. It supports Web Services of the "RPC" style (see note 2 below) that use the default SOAP encoding (see note 3) which is defined at http://schemas.xmlsoap.org/soap/encoding. The only methods that can be invoked are those which return data of the type typically called "xsd:string", or a fully compatible type. For more on the primitive data types, see the W3C Recommendation at http://www.w3.org/TR/xmlschema-2/.

The SOAPCall statement requires a minimum of 8 parameters. In addition, if the method (or operation) being invoked requires parameters, the SOAPCall statement will need to be given names and values for each. The following is a list of the SOAPCall parameters and a description of each, including where to find them in a typical Web Services Description Language (WSDL) file.

- ContinueOnError: A Boolean value indicating whether the simulation should continue processing if an error occurs. If false, an error terminates the simulation; if true, an error will be ignored. It's generally wise to use a false value at least until debugging has been completed. The parameter used can be a BOOLEAN or a STRING, as defined in the SIMPROCESS User's Manual. When a string is passed, any value (when converted to lowercase letters) other than "true" will be considered false.
- Endpoint URL: A string containing the URL of the endpoint for the Web Service to be invoked.
- Service Name: A string containing the Service Name to be invoked, as given in the WSDL file. In a typical WSDL file, there will be a <service> element (it may optionally be namespace qualified; see note 1) near the end of the WSDL file with a "name" attribute that will provide the value to use here.
- Service Port Type Name: A string containing the name of the service's port type. This will appear in a <port> element that is a child of the <service> element above, and the "name" attribute there will provide the service port name.
- Body namespace URI: A string containing the URI used for the body namespace. Note that this is a URI, not a URL. A URI can be a URL, or it can be an identifier of another sort. It could be a string such as "urn:foo", or perhaps "dispatcher:wsdl" (as is the case in our DispatcherService WSDL file). In a typical WSDL file it can be found on the root element's "targetNamespace" attribute, or the "xmlns:tns" attribute, often in both. Though every WSDL will likely provide a value, it will not always be critical to successfully invoking the service method, however. It's best to provide it just to be certain.

- Method namespace URI: A string containing the namespace URI of the method (operation) to be invoked. The <port> element mentioned above (see the Service Port Type Name parameter) will include a “binding” attribute providing a namespace qualified name for a binding (though most WSDL files will only have a single binding, service and port). The namespace may or may not be relevant, but the binding name found there will appear in the "name" attribute of a <binding> element (see notes 1 and 2). The child elements of the <binding> element will be <operation> elements whose "name" attribute values correspond to the available methods. Each <operation> element will have as children a <soap:operation> element (see the next parameter) as well as a pair of <input> and <output> elements. The <input> and <output> elements (the information on both should be identical, so choose one) each contain a <soap:body> child element. (See notes 3 and 4 for other important information about the <soap:body> element.) This will contain a “namespace” attribute whose value will be the method namespace URI to be used for this parameter to SOAPCall.
- SOAP Action URI: A string containing the SOAP Action URI. The <soap:operation> element described above contains a single attribute named “soapAction”. Its value will very often be an empty string, though not always (see note 5). Whatever value is found in that attribute should be provided for this parameter.
- Method (operation) to invoke: A string containing the name of the method (operation) to invoke.
- Additional parameters (optional): If the method to be invoked requires passing parameters (as most do), the remaining parameters provided to the SOAPCall statement must be in matched pairs. Each pair must include a string containing the name of the parameter, and a value to be passed for the named parameter at invocation.

    a. Parameter name: Refer to the <binding> element mentioned above. It will contain a “type” attribute providing the namespace qualified name for a "port type". This will refer to a <portType> element with a “name” attribute whose value is the same. The child elements of <portType> will be <operation> elements (one for each available operation) containing two key pieces of information:

        i. The value of the "name" attribute corresponds to the name of the method (operation) to be invoked.

        ii. The value of the “parameterOrder” attribute will list the names of all parameters that must be passed (the values to be included here in the SOAPCall statement), and the order in which they must appear.

    b. Parameter types: Although SOAPCall can only invoke operations returning the “xsd:string” (or fully equivalent) type as stated above, it can invoke operations whose parameters are functionally compatible with the SIMPROCESS types INTEGER (xsd:int), REAL (xsd:double), STRING (xsd:string) or BOOLEAN (xsd:boolean) as defined in the SIMPROCESS User's Manual. Internally, SOAPCall checks the parameters passed to ensure that they are of the allowable SIMPROCESS types. It then

assumes that the operation's named parameter is of the corresponding SOAP type. Only testing the SOAPCall statement with a specific operation will determine whether it can be used. (See note 6 for more information.)

Notes:

1. Depending on the developer of a Web Service and the toolset(s) used, the element names appearing in a WSDL file may be qualified with a namespace. For instance, the structure of a WSDL file is mandated by standard which is sometimes referred to via namespace, so that all the elements in a WSDL file may be qualified, like <wsdl:definitions> or <wsdl:operation>. When this approach is used, there will be a namespace reference in an attribute on the root element of the WSDL file, such as xmlns:wsdl="http://schemas.xmlsoap.org/wsdl", indicating the namespace prefix (e.g., “wsdl”) and the URI of the namespace. Much of the above information is based on the assumption that WSDL file elements are not namespace qualified, as is true of SIMPROCESS' DispatcherService.

2. A typical <binding> element (optionally namespace qualified; see note 1) will be immediately followed by a <soap:binding> element with a “style” attribute. SOAPCall only works with those having a value of “rpc” here; if the style is “document” the SOAPCall statement cannot be used with this service.

3. The <soap:body> element within each <operation> element (optionally namespace qualified, as per note 1) should contain an attribute named "use". This attribute should have a value of “encoded”; if its value is “literal”, the SOAPCall statement cannot be used with this service.

4. The <soap:body> element within each <operation> element (optionally namespace qualified; see note 1) should contain an attribute named “encodingStyle”. Its value should be http://schemas.xmlsoap.org/soap/encoding/ which defines the standard encoding for SOAP 1.1. If any other URL is used, it may not be possible to use SOAPCall with this service.

5. Many services will indicate that the SOAP Action URI is only meaningful if the transport is HTTP. Because SIMPROCESS is a Java application, it uses HTTP transport automatically in most environments, so this value should always match that found in the WSDL file. If it isn't needed, the environment will transparently discard it.

6. There is no limit to the type of information that can be exchanged between client applications and Web Services using SOAP. There is, however, a practical limitation to how the SOAPCall statement can be prepared in advance to interact with Web Services chosen by SIMPROCESS users. Internally, SIMPROCESS uses what is known as the Dynamic Invocation Interface to build and invoke the call to a Web Service operation. Other available approaches require prior knowledge of the WSDL file and require generation and compilation of Java code to call it. This simply isn't possible in a runtime environment. As a result, we've had to select the best all around means of offering access to Web Services via SOAP. In the future, as available tools for runtime examination of WSDL files improve, perhaps SIMPROCESS will be able to improve the SOAPCall statement to provide still greater capabilities.

Method InvokeWebService

The InvokeWebService statement allows SIMPROCESS models to invoke operations (or methods) on Web Services. The older SOAPCall statement should now only be used for a Web Service using the older "rpc/encoded" form, as newer technologies no longer support this type of operation.

The InvokeWebService statement requires a minimum of 5 parameters. In addition, if the operation being invoked expects parameters, values should be provided for each. The following is a list of the InvokeWebService parameters and a description of each, including where to find them in a typical Web Services Description Language (WSDL) file.

- ContinueOnError: A BOOLEAN value indicating whether the simulation should continue processing if an error occurs. If false, an error terminates the simulation; if true, an error will be ignored. It's generally wise to use a false value at least until debugging has been completed. The parameter used can be a BOOLEAN or a STRING, as defined in the SIMPROCESS User's Manual. When a string is passed, any value (when converted to lowercase letters) other than "true" will be considered false. There are certain conditions which will result in an error regardless of this setting. See Note 1 for additional information.
- WSDL URL: A string containing a URL where the WSDL for the Web Service can be obtained.
- Service Name: A string containing the Service Name, as given in the WSDL file. In a typical WSDL file, there will be a <service> element (it may optionally be namespace qualified; see Note 2) near the end of the WSDL file with a "name" attribute that will provide the value to use here.
- Service Port Type Name: A string containing the name of the service's port type. This will appear in a <portType> element (also optionally namespace qualified), and its "name" attribute will provide the service port name.
- Operation (method) to invoke: A string containing the name of the operation to invoke. A typical WSDL file will have one or more <operation> elements as children of <portType>, each having a "name" attribute with the operation's name.
- Additional parameters (as required by the operation to be invoked): If the operation expects parameters, the remaining arguments provided to the InvokeWebService statement must match those in both number and type. A WSDL file may include a reference to an external XML Schema file describing all the parameters, or it may be included "in-line" within the WSDL itself, but this schema will be the source of information regarding number and types of parameters expected. The invocation of a Web Service operation will be done by generating and compiling Java code using the other parameters to InvokeWebService, then evaluating this list of parameter values against what's expected by the operation. Here are some general guidelines:
  - A parameter of type "string" corresponds to the STRING type in SIMPROCESS.
  - The INTEGER type in SIMPROCESS is compatible with the Java primitive type "int" or with Integer.

- The REAL type in SIMPROCESS is compatible with the Java primitive type "double" or with Double.
- The BOOLEAN type in SIMPROCESS is compatible with the Java primitive type "boolean" or with Boolean.
- Any parameter for which you need to provide a Java "null" value should be passed as a STRING with "null" (including the quotes) as its value. This will result in an error, however, if the method's parameter is a Java primitive type (i.e., boolean, int or double) since those are not reference data types in Java.
- Any value will be accepted if it is "assignment compatible" with the Java type indicated for the method parameter. For example, a value can be passed whose type might be a subclass of the method's parameter.
- Any parameter which is not one of the predefined types must be passed using a local variable or Attribute of type ANYOBJ. It may be necessary to devise external Java code which can be called via the ExternalCall statement to construct and/or convert values for this purpose. For example, the setModelParameters operation in the SIMPROCESS DispatcherService expects to receive a List<String> with the model parameters to be set. Since SIMPROCESS cannot create this type of structure, an array made using CreateArray can be populated, then converted with external Java code and stored in an appropriate location for subsequent use as an operation parameter.

The InvokeWebService statement will always return a value. If the operation being invoked does not return anything (i.e., it represents a Java "void" method), then a BOOLEAN value of TRUE will result upon successful invocation. Otherwise, InvokeWebService will return a value as specified in the WSDL and XML Schema. An error will result if the type returned does not match the variable to which it is being assigned.

Notes:

1. Some situations may arise when attempting to generate, compile and execute Java code which will result in errors regardless of the ContinueOnError setting. Some examples include:
   - Failure to create a temporary directory into which code can be generated.
   - Failure to generate Java code for the Web Service. This could result from being unable to access the WSDL at the provided URL, from a network failure, or any of a variety of other causes. There will often be additional information available in the error file.
   - Failure to compile the generated Java code. There will often be additional information available in the error file.
   - Certain errors occurring when executing the generated and compiled Java code are also fatal. These include being unable to obtain a ClassLoader to access the generated code, or being unable to obtain an instance of the class representing the Service Name or Service Port Type Name provided. Most other errors will respond according to ContinueOnError.

2. Depending on the developer of a Web Service and the toolset(s) used, the element names appearing in a WSDL file may be qualified with a namespace. For instance, the structure of a WSDL file is mandated by standard which is sometimes referred to via namespace, so that all the elements in a WSDL file may be qualified, like <wsdl:service> or <wsdl:operation>. When this approach is used, there will be a namespace reference in an attribute on the root element of the WSDL file, such as xmlns:wsdl="http://schemas.xmlsoap.org/wsdl", indicating the namespace prefix (e.g., "wsdl") and the URI of the namespace. Much of the above information is based on the assumption that WSDL file elements are not namespace qualified, as is true of SIMPROCESS' DispatcherService.

Methods CreateMap, PutInMap, GetFromMap, RemoveFromMap, ClearMap, GetMapKey, GetMapSize

These methods are used to create and manipulate hashmaps. **`CreateMap`** creates a hashmap that stores REAL, INTEGER, BOOLEAN, STRING, or ANYOBJ values. **PutInMap** places a keyed value into a hashmap. The key can be of type REAL, INTEGER, BOOLEAN, STRING, or ANYOBJ. The value must match the type used in **CreateMap**. **GetFromMap** retrieves a value from a hashmap without removing the value from the hashmap. **RemoveFromMap** retrieves a value from a hashmap and removes the value and its key from the hashmap. **ClearMap** removes all values from the hashmap along with their associated keys. **GetMapSize** and **GetMapKey** are usually used when iterating through a map. An Attribute with a **Mode** of Object must be defined to create a hashmap. For example, assume there is a Model Attribute named Map with **Mode** Object (with **Do Not Reset Before Each Replication** selected). The following statement should go in the **`Start Run`** Expression of the **Model Expressions**.

```
Model.Map := CreateMap("ANYOBJ");
```

This creates a hashmap that will store any type of object. Use **PutInMap** to place values in the hashmap. For instance assume that there are several activities that will need to be referenced throughout the model. Certain decision points will required knowledge of how many entities are currently in these activities. The following could go into the **`Start Simulation`** expression of each activity.

```
PutInMap(Model.Map, Name, Self);
```

This places each activity into the hashmap with the name of the activity as the key to the value (the activity itself). Note that the keys must be unique. Thus, if two activities being placed into the hashmap have identical names, only the last one placed in the hashmap will be there. The last one put in the map will replace the previous entry.

Assume in the **`Accept Entity`** expression of a Branch Activity we need to determine the appropriate path for the incoming Entity based on information from the Activities in the hashmap. In this example there are two Activities in the hashmap: **`Call Wait Queue`** and **`Calls In Service`**. GetFromMap can be used to retrieve each Activity and check each Activity's status.

```
callWait : ANYOBJ;
```

```
callService : ANYOBJ;
callWait := GetFromMap("Call Wait Queue");
callService := GetFromMap("Calls In Service");
IF callWait.NumberIn > 100 AND callService.NumberIn > 25
   Entity.Reject := TRUE;
END IF;
```

RemoveFromMap also returns the value stored in the hashmap, but it also removes the value from the map. Thus if the statements below were used, the next entity entering the Branch Activity would get an error since the Activities no longer exist in the hashmap.

```
callWait := RemoveFromMap("Call Wait Queue");
callService := RemoveFromMap("Calls In Service");
```

RemoveFromMap should be used when the values in the hashmap are temporary in nature, such as Entities. If temporary objects are placed into a hashmap, it is important that they be removed before being disposed. If not, memory used by the temporary object will not be freed.

ClearMap should normally be used in the End Run expression. It ensures that unneeded memory is freed at the end of the simulation.

```
ClearMap(Model.Map);
```

GetMapSize returns the number of objects in a hashmap. This number can be used to iterate through the hashmap using **GetMapKey** in a **FOR** loop. The type (INTEGER, REAL, etc.) of the variable that will contain the key returned by **GetMapKey** must match the type of the key returned. (Thus, it is recommended that all the keys be of the same type for a particular hashmap.) The example below shows iterating through a hashmap that contains Entities keyed by their **SequenceNum**. Items should not be added to or removed from a hashmap when iterating through a hashmap.

```
size : INTEGER;
count : INTEGER;
key : INTEGER;
entity : ANYOBJ;
size := GetMapSize(Model.Map);
FOR count := 1 TO size
   key := GetMapKey(Model.Map, count);
   entity := GetFromMap(Model.Map, key);

   {Alternatively, the above two lines could be replaced with
   entity := GetFromMap(Model.Map, GetMapKey(Model.Map, count));}

   IF entity.HoldTime > 200.0
      entity.Reject := TRUE;
   END IF;
```

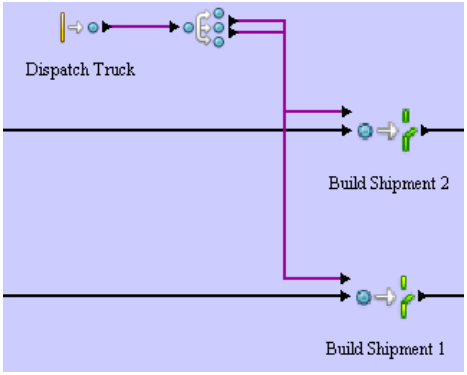
```
END FOR;
```

Method BREAKPOINT

BREAKPOINT assists with model debugging. There are many instances when it may be useful to pause the model to check the status of the simulation, but primarily **BREAKPOINT** statements can be used to

- Catch model anomalies
- Verify Expression operation

The first example below, used to explain the dialog tabs, shows using a **BREAKPOINT** statement to catch a model anomaly. In this particular portion of the model, order fulfillments are held in Gate Activities until a truck is available. Trucks leave every 12 hours and there should always be fulfilled orders waiting for shipment. A problem with the model is indicated if a truck arrives and there are no orders waiting to be shipped.



The truck arriving is modeled by an Entity that is generated every 12 hours. This Entity and its clone are sent to the Trigger pads of the Gate Activities. Since the Gate Activity ignores a trigger if there are no Entities in the Gate, SIMPROCESS will not indicate that there is a problem with the model (that is, there are no orders waiting to be shipped when the truck arrives). However, placing the following Expression is in the Accept Entity Expression of each Gate Activity allows this problem to be immediately indentified if it occurs.

```
IF ActivatedByTrigger AND NumberIn = 1
  BREAKPOINT;
END IF;
```

This Expression checks to see if the Entity entering the Gate is entering by the Trigger pad and checks to see if the number of Entities in the Gate is only one. Both being true (the Entity entering is the Trigger

Entity and it causes **NumberIn** to be one) indicates that there are no fulfilled orders in the Gate Activity. Thus, **BREAKPOINT** is executed which causes the simulation to pause. While paused the status of any item in the model can be examined in order to determine why there were no fulfilled orders when a truck arrived. For example, the Activities prior to the Gate Activities can be examined for bottlenecks or incorrect Entity routing. If there appears to be a bottleneck, any Resources required at the suspect Activity can be viewed to determine if the Resource levels are appropriate or if the actual last delay time is appropriate. Bottom line is the BREAKPOINT statement not only shortens the time needed to identify the problem but also shortens the time needed to find the cause of the problem and fix it.

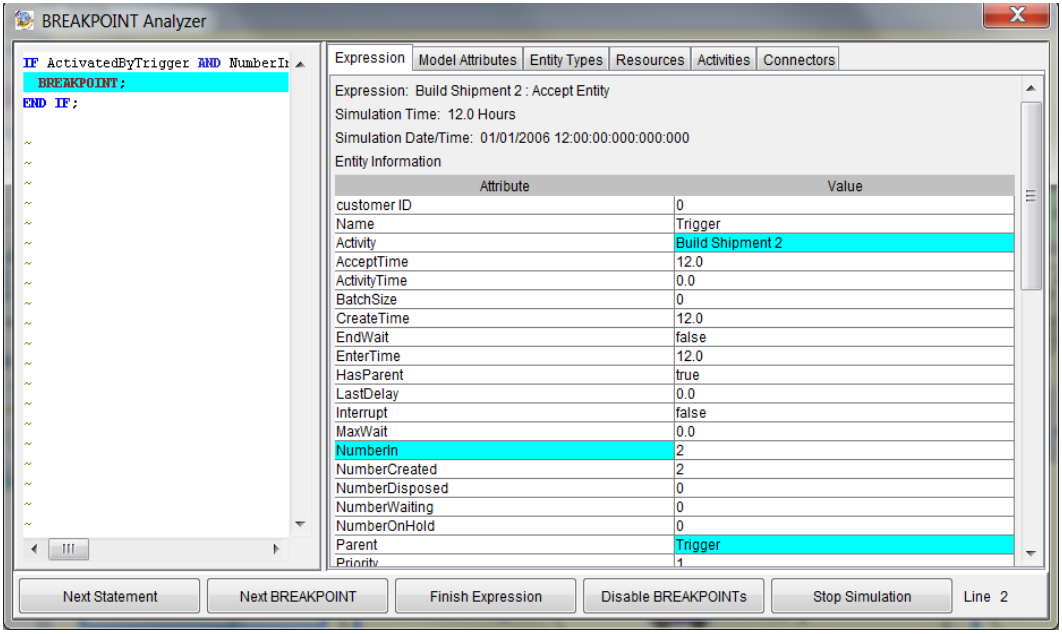
The second example, which is used to explain the dialog buttons, shows how **BREAKPOINT** statements can be useful in verifying Expressions. The longer the Expression the more likely there are errors, but even short Expressions can have unrecognized errors. That is, the Expression may execute without any runtime errors but due to incorrect logic, calculations, variables, or attributes the results are not correct. This model (**ConfigureWithDatabase.spm** in **ExpressionDemos** directory) has a complex Expression that runs when the simulation starts that reads in data from a database. For this example a **BREAKPOINT** statement was placed in the inner **WHILE** loop of an outer **WHILE** loop. By including a **BREAKPOINT** statement at the end of the inner **WHILE** loop, the simulation pauses at the end of the inner loop so that as the data is being read in it can be verified. This example only shows one **BREAKPOINT** statement in an Expression, but there can be multiple **BREAKPOINT** statements within the same Expression.

```
{Get the different types of technicians}
sql := "Select * From TechnicianTypes;";
ReadFromDatabase(Model.Database, "Set", sql);
WHILE GetNext("Set")
  {Count the number of technician types}
  Model.NumTechnicianTypes := Model.NumTechnicianTypes + 1;
  name := GetResult("Set", "TechnicianType");
  id := GetResult("Set", "TechnicianId");
 {Get the number of each type of technician by querying the Employees
  table and counting the number of employees for each technician type.}
  sql := "Select Count(EmployeeId) As EmployeeCount From Employees
          Where TechnicianId=" + INTTOSTR(id) + ";";
  ReadFromDatabase(Model.Database, "EmpSet", sql);
  IF GetNext("EmpSet")
      {Resource units will be the number of the employees whose
       TechnicianId matches the current TechnicianType}
    units := GetResult("EmpSet", "EmployeeCount");
  ELSE
    {End simulation with error}
```

```
    HALT("No Employees for technician type " + name + ".");
  END IF;
  {Create the resource with the selected name and units}
  CreateResource(name, FLOAT(units), FALSE, FALSE, FALSE, FALSE);
  {Set the downtime to a predefined global downtime}
  SetResourceDowntime(name, "Technician Downtime");
  {Set the hourly cost}
  cost := GetResult("Set", "HourlyCost");
  SetResourceCost(name, cost, "PerTimeUnit", "Hours");
  {Set up expressions for the Resources}
  sql := "Select ExpressionId From ExpressionUse Where
         ExpressionOwnerId=" + INTTOSTR(id) + ";";
  ReadFromDatabase(Model.Database, "ExpUseSet", sql);
  WHILE GetNext("ExpUseSet")
    expressionId := GetResult("ExpUseSet", "ExpressionId");
    IF expressionId > 0
      {Get the expression and set it in the Resource}
      sql := "Select * From Expressions Where ExpressionId=" +
             INTTOSTR(expressionId) + ";";
      ReadFromDatabase(Model.Database, "ExpSet", sql);
      IF GetNext("ExpSet")
        expressionType := GetResult("ExpSet", "ExpressionType");
        expressionText := GetResult("ExpSet", "ExpressionText");
        SetResourceExpression(name,expressionType,expressionText);
      END IF;
    END IF;
    BREAKPOINT;
  END WHILE;
  techName := "TechnicianType" + INTTOSTR(id);
  {Put the name of the Resource into a map keyed by TechnicianType1,
  2, etc.}
  PutInMap(Model.TechnicianNameMap, techName, name);
END WHILE;
```

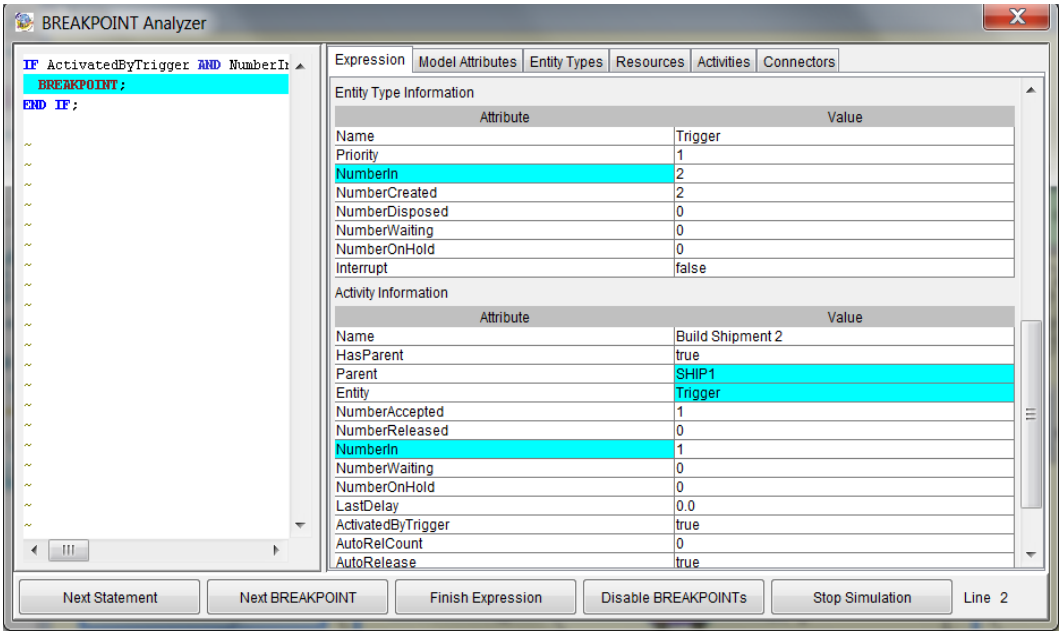
When a **BREAKPOINT** statement is encountered in an Expression and **BREAKPOINT** statements have been activated (see "Expression Options"), the simulation pauses and the **BREAKPOINT Analyzer** dialog

displays.

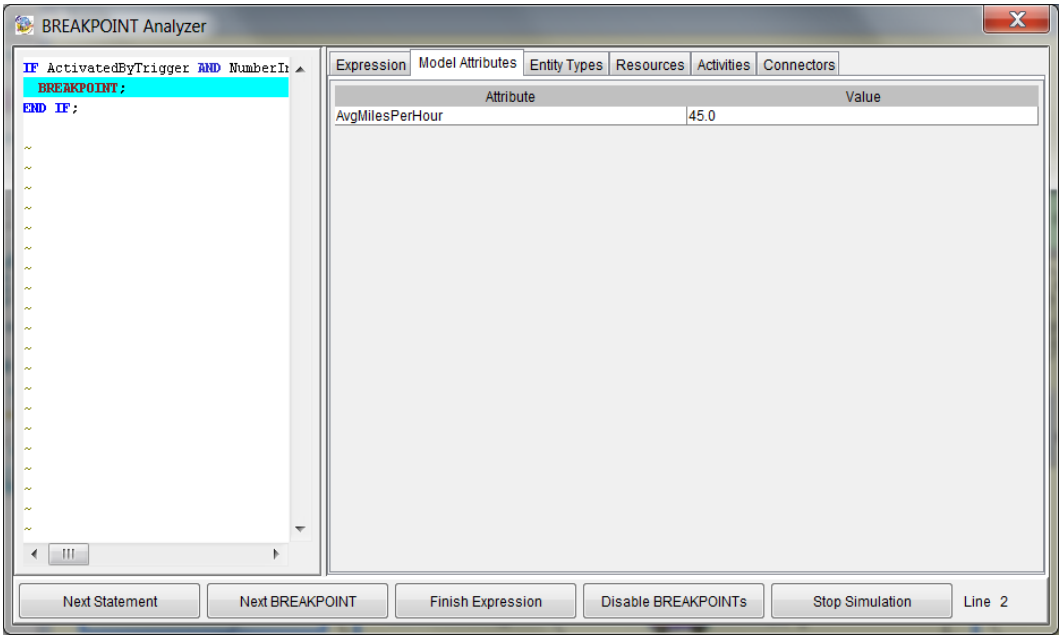


On the left is the Expression with the active **`BREAKPOINT`** statement highlighted. On the right are six tabs: **Expression**, **Model Attributes**, **Entity Types**, **Resources**, **Activities**, and **Connectors**. The line number of the highlighted **`BREAKPOINT`** statement is displayed at the bottom right. The divider between the two sides can be moved by clicking on the divider and holding the mouse button down while dragging to the left or right. Also, for all tables, the column widths can be adjusted by clicking and dragging the table header just above where the column divider exists in the table. The mouse should change to indicate dragging is allowed. Double clicking on highlighted items will open a dialog either displaying detail for the highlighted item, or a list of Entities if the highlighted item represents a queue.

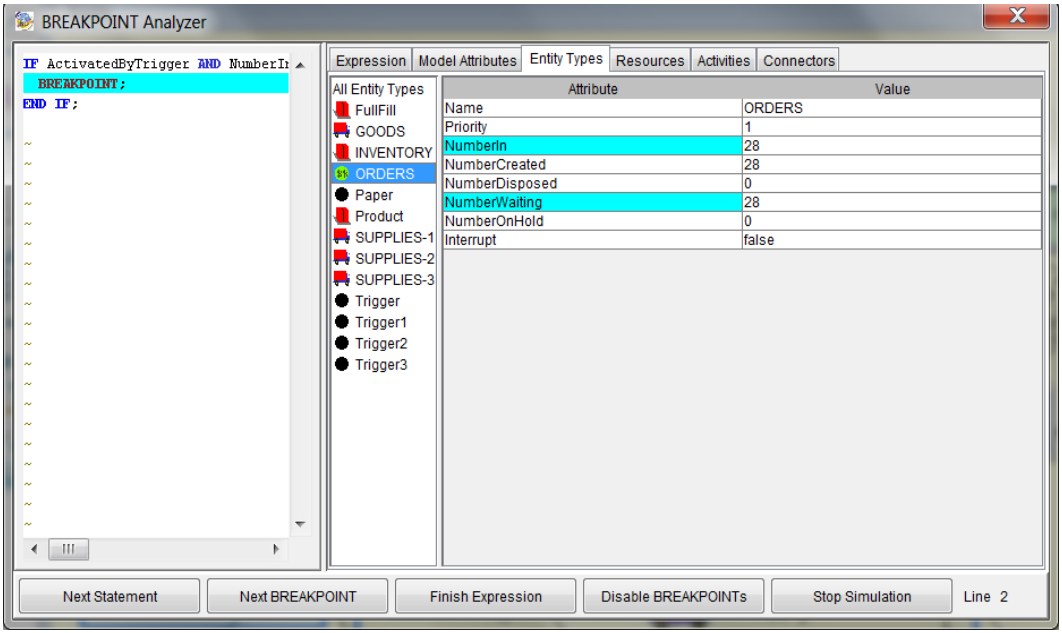
The **Expression** tab gives the name of the expression, the current simulation time, the current simulation date/time, and Expression information. The Expression information contains the status of any local variables defined before the **`BREAKPOINT`** statement and any Entity, Entity Type, Activity, or Resource information associated with the expression. In the example above, since there are no local variables defined and since it is an Activity expression, there is information for the Entity instance, the Entity Type, and the Activity. (The Entity Type and Activity information is shown below.) The tables include all user-defined attributes and system attributes for each item. The example shows one user-defined attribute for Entity instances: **`customerID`**. The remaining visible attributes are Entity system attributes.



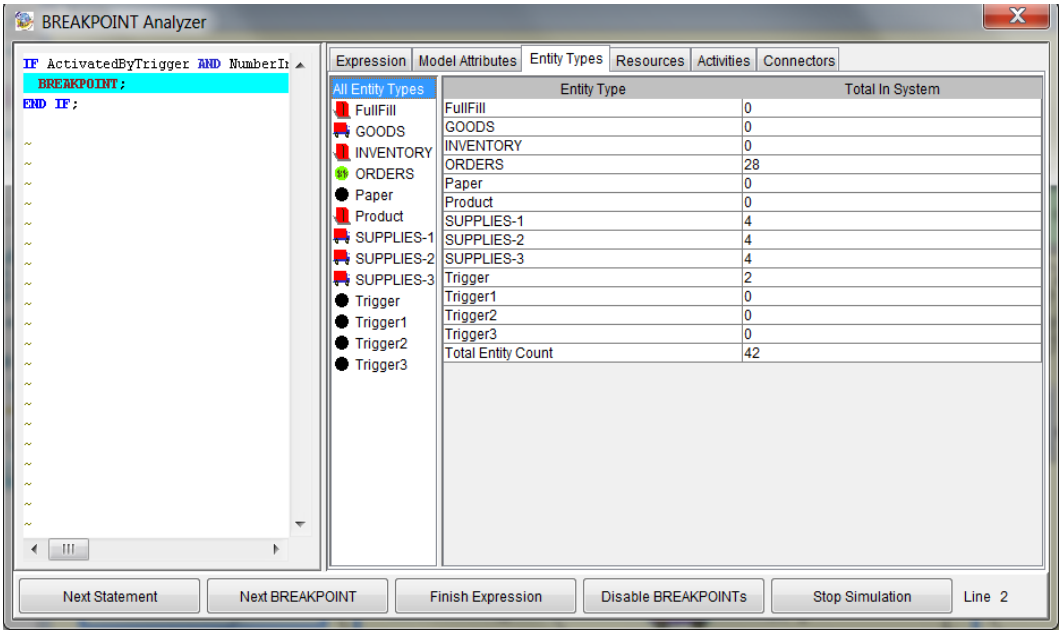
The Model Attributes tab displays the status of all model attributes.



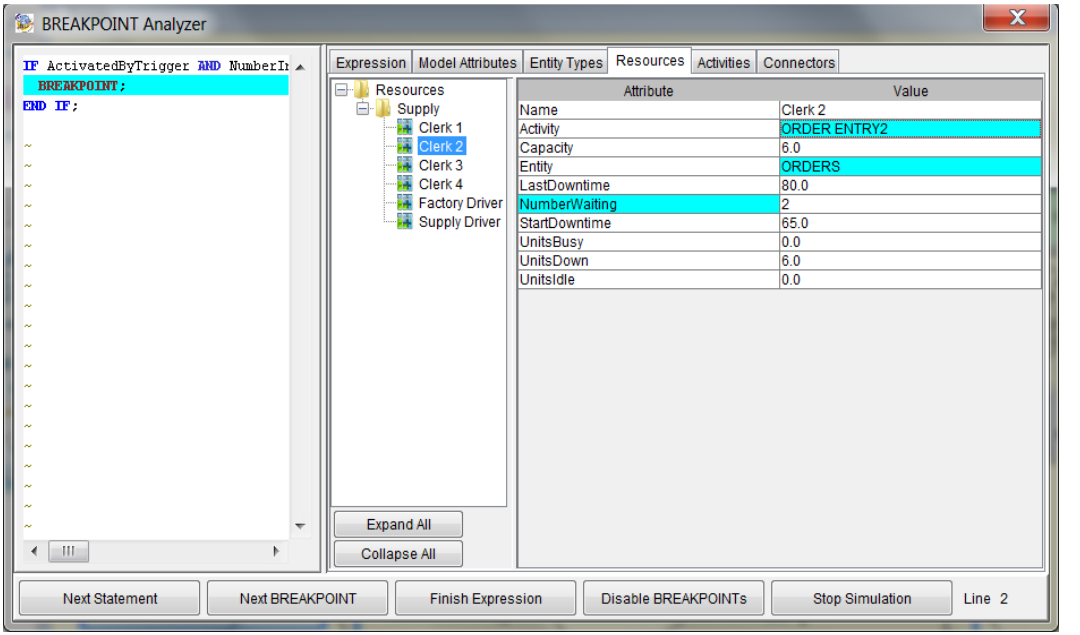
The **Entity Types** tab lists all the Entity Types defined in the model along with an extra entry called **All Entity Types**. Selecting an Entity Type displays all the user-defined and system attributes for that selection. Note that this tab and all remaining tabs contain a divider between the items to select on the left and the table with the selected information on the right. Again, as with the main divider, it can be adjusted left or right by clicking and dragging.



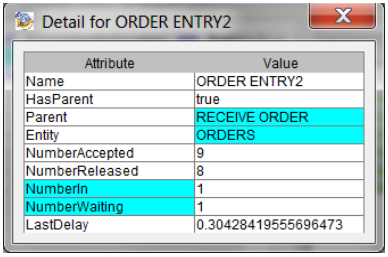
All Entity Types lists all the defined Entity Types along with the number of instances of each type currently in the simulation. At the end it gives the total number of Entities in the simulation.



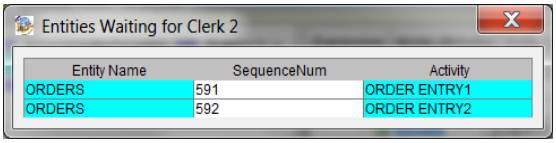
The **Resources** tab displays the model Resource tree. Selecting a Resource displays the user-defined and system attributes for the selected Resource.



Double clicking a highlighted item will display additional detail. In the example above, double clicking ORDER ENTRY 2 will display the detail for that Activity.

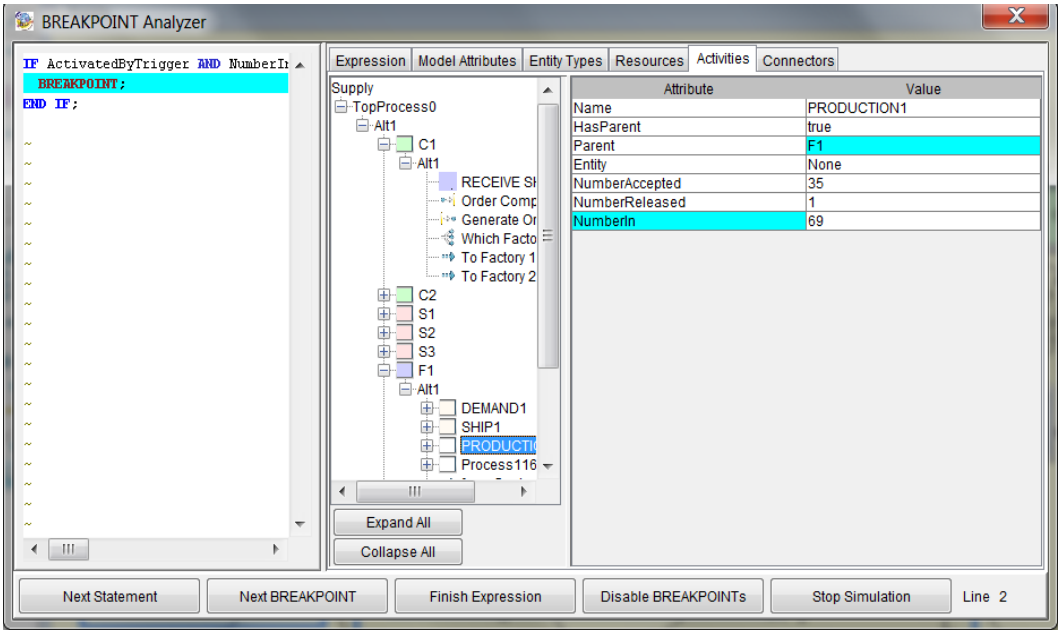


The detail dialog also has highlighted items that can be double clicked. For example, double clicking NumberIn will display a list of the Entities in the Activity ORDER ENTRY 2. Double clicking NumberWaiting will display a list of the Entities waiting for a Resource at ORDER ENTRY 2. If the above dialog is closed and NumberWaiting is double clicked on the Resources tab, a list of all Entities waiting for Clerk 2 is displayed.



Clicking a column header will sort the table in ascending order based on the values in that column. Holding the Shift key when clicking a column header will sort the table in descending order based on the values in that column.

The Activities tab display the model Activity tree, and the Connectors tab displays the model Connector tree. Again, selecting an item in the tree displays the items attributes.

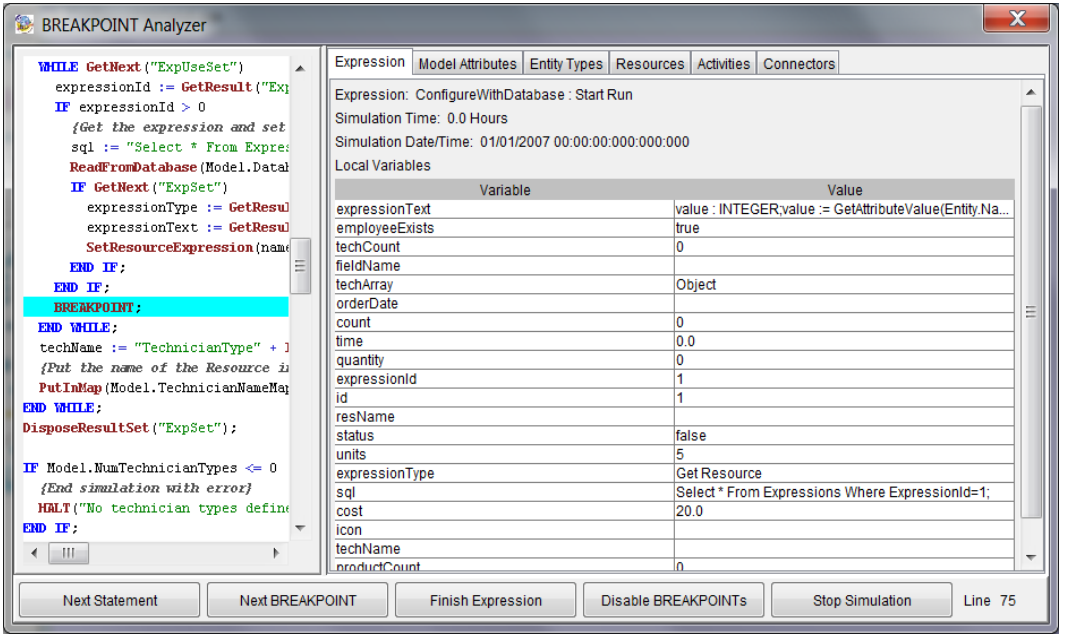


The buttons at the bottom of the dialog are used to control **`BREAKPOINT`** operation or manipulate item trees (**Resources**, **Activities**, and **Connectors** tabs only). The **Next Statement** button moves to the next statement in the Expression. (Comments and blank lines are ignored.) All tabs are updated. This button works correctly only when the following conditions are true.

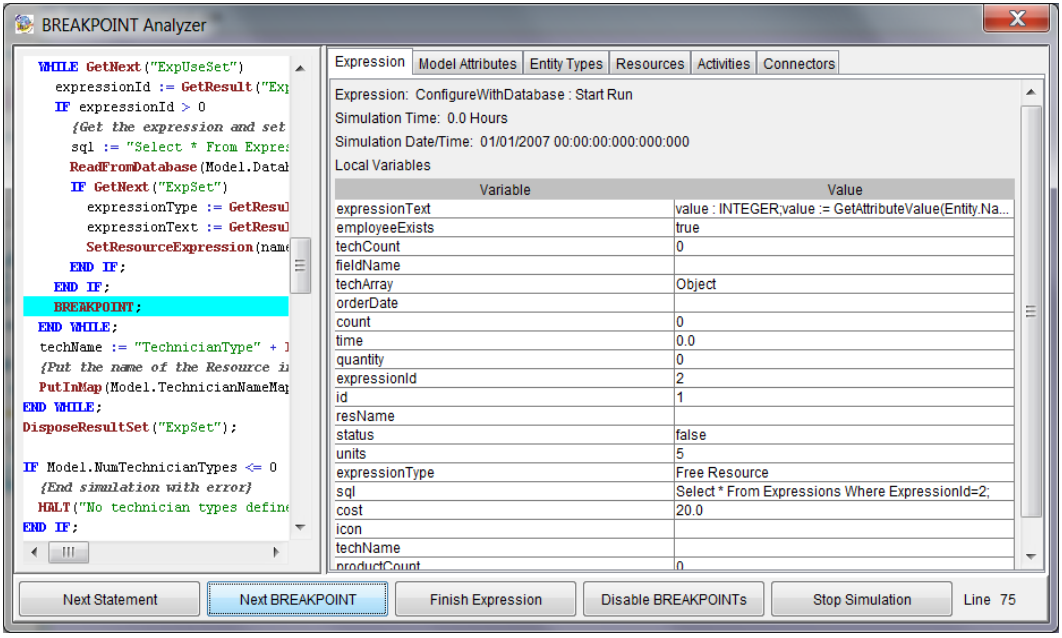
- There is only one statement on each line of the Expression.
- Each statement is fully contained on one line of the Expression.
- A line with an active statement cannot include a comment.

The **Next BREAKPOINT** button moves to the next **`BREAKPOINT`** statement in the Expression. All tabs

are updated. If there are no more **BREAKPOINT** statements in the Expression the dialog closes and the simulation resumes. In the example below the simulation has paused at the end of an inner **WHILE** loop.



Since the **BREAKPOINT** statement is within two loops, clicking **Next BREAKPOINT** will cause the loops to execute until the **BREAKPOINT** statement is reached again. Note that by clicking **Next BREAKPOINT** once the value of the local variable **expressionId** changes from 1 to 2 and the value of the local variable **expressionType** changes from **Get Resource** to **Free Resource**. The value of the local variable **sql** also changed but the change is not visible without adjusting the **Value** column.



The **Finish Expression** button resumes execution of the Expression but ignores all remaining **`BREAKPOINT`** statements in the Expression. This button causes the dialog to close and resumes the simulation.

The **Disable BREAKPOINTs** button closes the dialog and disables all **`BREAKPOINT`** statements in the model so the simulation runs without interruption. This button does not change the setting of **Enable BREAKPOINT Statements** on the **Expression Options** tab of the Run Settings. Thus, if the model is run again, the **`BREAKPOINT`** statements are active unless **Enable BREAKPOINT Statements** is deselected. Once the **`BREAKPOINT`** statements have been disabled there is no way to reactivate them during the current run.

The **Stop Simulation** button closes the dialog and stops the simulation.

The **Expand All** and **Collapse All** buttons are available whenever the **Resources**, **Activities**, or **Connectors** tab is selected. These buttons operate on the item trees on each tab.

Method InterruptEntity

Interrupts the Entity specified by **`SequenceNum`**. **`SequenceNum`** is the unique id number of the Entity. The interrupted Entity is sent to the specified Local Receive Transfer Activity. (See "Transfer Activity" for information on the Transfer Activity.) Returns TRUE if the interrupt is successful. (See

InterruptDemo.spm in the **ExpressionDemos** directory.)

If the Entity is processing at an activity when the interrupt occurs, any Resources obtained at the Activity are released. Resources obtained at a Get Resource Activity or obtained using the **GetResource** expression statement without a corresponding **FreeResource** statement are not released.

If the Entity is waiting for Resources when the interrupt occurs, the Entity is removed from the wait queue of each Resource before being sent to the Transfer Activity.

If the Entity is holding for a condition in an Assemble, Batch, Gate, Join, or Synchronize Activity when the interrupt occurs the Entity is simply removed from the hold and sent to the Transfer Activity. An Entity waiting at a Join Activity that is interrupted loses its Family reference. Thus, it is no longer considered when determining whether all family members are present at a Join activity. Note, though, that any cloned Entity that is interrupted does not lose its reference to its Parent Entity. (See "Split and Join Activities" for more information.)

A runtime error will occur if **InterruptEntity** is used on an Entity that is part of a batch or is already interrupted due to Resource downtime (see "Defining Downtime Schedules of Resources") or Entity preemption (see "Defining an Entity"). Entities delaying on a Connector cannot be interrupted. However, no runtime error will occur. FALSE will be returned.

Note that the Entity System Attributes **InBatch**, **Processing**, **WaitingForResource**, and **HoldingForCondition** can be used to determine the state of an Entity before using **InterruptEntity**. (See "System Attributes".)

This method requires knowing the **SequenceNum** of a specific Entity so if there is potential for interrupting an Entity there must be a way to keep track of the **SequenceNum**. This can be done using Attributes, Arrays, Maps, or parent/child relationships. For example, assume a clone Entity is created at a Split Activity and the original Entity may need to interrupt the clone. The **SequenceNum** of the clone can be saved in an Attribute on the original entity. Alternatively, if the clone needs to interrupt the original then **Entity.Parent.SequenceNum** can be used.

```
status : BOOLEAN;
status := InterruptEntity(Entity.Parent.SequenceNum, "Transfer1");
```

If less is known about the Entity to be interrupted it may be necessary to check conditions.

```
entity : ANYOBJ;
status : BOOLEAN;
entity := GetEntityBySequenceNum(42);
IF (entity <> UNDEFINED) AND (NOT entity.InBatch)
  IF entity.Processing
    status := InterruptEntity(42, "Transfer1");
  END IF;
END IF;
```

Method InterruptActivity

Interrupts the Activity (or Activities if multiple Activities have the same name) specified by Activity Name. If no Entity State parameters are specified or an Entity State parameter is "All", Entities processing, waiting for Resources, or holding for a condition are interrupted. The Receive Transform parameter is the name of the Local Receive Transform Activity that will receive the interrupted Entities. (See "Transfer Activity" for information on the Transfer Activity.) Returns the number of Entities interrupted. (See **`InterruptDemo.spm`** in the **`ExpressionDemos`** directory.)

Entities are interrupted according to the same rules as **`InterruptEntity`**. However, there are no runtime errors due to Entities already interrupted because of Resource downtime or Entity preemption. Entities in these states are simply ignored.

Entities are interrupted in the reverse order that they entered the Activity. This is so Entities waiting for Resources do not attempt to obtain Resources when those Entities processing with Resources are interrupted. Thus, Entities waiting for Resources are interrupted before Entities that are processing.

Up to two Entity State parameters may be specified. Allowable values are "All", "Processing", "WaitingForResource", or "HoldingForCondition". If no Entity State parameters are specified or an Entity State parameter is "All", Entities in all states (processing, waiting for Resources, or holding for a condition) are interrupted.

```
count : INTEGER;
count := InterruptActivity("Delay5", "Transfer12");
```

Specifying an Entity State does not produce an error if the Activity being interrupted does not cause an Entity to enter that state. For example

```
count : INTEGER;
count := InterruptActivity("Delay5", "Transfer12",
"HoldingForCondition");
```

does not cause an error for a Delay Activity even though Delay Activities do not have the state "Hold For Condition".

If the specified Activity is a Process, all Activities within the Process are interrupted including Activities within subprocesses. There is no way to set the interrupt order of the Activities in the Process; thus, if interrupt order is important, multiple **`InterruptActivity`** statements should be used.

SIMPROCESS Color Table

0	Black	33	SandyBrown
1	White	34	Gold
2	Red	35	Goldenrod
3	Green	36	LightGrey
4	Blue	37	MediumGoldenrod
5	Yellow	38	DarkOliveGreen
6	Cyan	39	ForestGreen
7	Magenta	40	LimeGreen
8	DarkGrey	41	MediumForestGreen
9	Grey	42	MediumSpringGreen
10	DarkRed	43	PaleGreen
11	DarkGreen	44	SeaGreen
12	DarkBlue	45	YellowGreen
13	DarkYellow	46	DarkSlateGrey
14	DarkCyan	47	DimGrey
15	DarkMagenta	48	Khaki
16	Aquamarine	49	Maroon
17	MediumAquamarine	50	Orange
18	CadetBlue	51	Orchid
19	CornflowerBlue	52	DarkOrchid
20	DarkSlateBlue	53	Pink
21	LightBlue	54	Plum
22	LightSteelBlue	55	IndianRed
23	MediumBlue	56	OrangeRed
24	MediumSlateBlue	57	VioletRed
25	MidnightBlue	58	Sienna
26	NavyBlue	59	Tan
27	SkyBlue	60	Turquoise
28	SlateBlue	61	DarkTurquoise
29	SteelBlue	62	Violet
30	Coral	63	Wheat
31	Firebrick	64	GreenYellow
32	Brown		

APPENDIX G

External Event Files

The Generate activity offers a File schedule. This appendix describes the format for the event file associated with the File schedule.

General Rules for Event Files

External event files define entity generation events. If event files are referred to in a Generate activity, SIMPROCESS reads the file at the beginning of a simulation run and schedules the generation of Entities as defined in the file.

Each record (line) in an event file must adhere to certain syntax rules, which are defined in the following section. Records must also adhere to these general rules:

- Each record in the file defines one generation event. There is no way to continue a record on the following line.
- The keywords (ENTITY, QUANTITY, etc.) followed by a colon (:) or equals sign (=) must not have any spaces between the colon or the equals sign and must not have any spaces between the colon or equals sign and the value associated with the keyword.
- The keywords and values are case sensitive.
- Records must be ordered by entity create time. SIMPROCESS will not generate entities for any event record whose entity create time is earlier than that of the previously processed event. For example, in the following sequence of creation dates:

    1. 01/02/2014
    2. 01/04/2014
    3. 09/11/2013
    4. 10/30/2013
    5. 01/12/2014

Records 1 and 2 are processed. Record 3 is bypassed, because its creation date is earlier than that of record 2. Record 4 is also bypassed, because its date is earlier than that of record 2, the last record for which entities were generated.

- Entity create time must be the first field in the event record. The remaining keyword parameters may be entered in any order.
- The entity type you specify in an event record must be defined to the model before it can be generated.

However, you can refer to a new type in the event file and select the **Selected in the list** option in the **External File Schedule** dialog. SIMPROCESS will automatically add it to the model.

- Any literal values with imbedded blanks must be enclosed in quotation marks (" ") when included in event records.
- Real numbers must include a decimal point.

- Records beginning with an asterisk (*) are considered comments and are ignored by SIMPROCESS.

Event Record Description

In the following syntax diagram:

- Items in **BOLD** face must be entered exactly as shown.
- Items in *italic* face represent variables for which you must substitute a value.
- Items enclosed in brackets ([]) are optional.

Syntax

[+]*time* [**QUANTITY:***value*] [**ENTITY:** *type*] [**ATTR:** *name*=*value*...]

Parameters

+

Indicates that the ensuing value is to be added to the time of the last event processed.

time

The time at which entities are to be created for this event record. Specify either a specific date and time or a relative value. Use the following format for a specific date and time:

mm/*dd*/*yyyy hh:mm:ss*

Where, for example, January 15, 2007, at noon is represented as:

01/15/2007 12:00:00

You can enter either real numbers or integers to indicate relative values: real numbers are assumed to refer to hours, while integers refer to days. Forty-eight hours after the start of simulation is shown as:

48.0

Which is the same as two days after the start of the simulation:

2

Forty-eight hours after the previous entity generation event:

`+48.0` **or** `+48:00` **or** `+2`

Notes on specifying event time:

- You can omit seconds when specifying the time. SIMPROCESS assumes a zero for the omitted parameter.
- Do not specify a time in *hh:mm* without a date, unless you precede it with a plus sign (+). Although SIMPROCESS will accept this value, its interpretation is unpredictable. For example, acceptable formats are:

  `48.0` 48 hours after the start of simulation

  `+48:00` 48 hours after previous generation event

- SIMPROCESS will accept a 2-digit value for the year, It assumes that values over 49 refer to years in the 20th century, and values under 50 refer to the 21st century.

QUANTITY:value

The number of entities to generate at the entity generation event. Enter a number in place of *value*.

If QUANTITY is omitted, SIMPROCESS uses the default **Quantity** value in the **Generate Activity Properties** dialog.

ENTITY: type

The type of entity to generate.

For *Type,* enter an entity type name. If the name includes blanks, enclose the entire string in quotes.

If ENTITY is not specified, SIMPROCESS uses the default Entity value in the **Generate Activity Properties** dialog.

ATTR: name=value...

Assigns values to globally defined attributes for entity instances.

You can assign values to any number of attributes in the VARS argument.

The *value* you specify must be consistent with the data type defined for the attribute.

Note that if you specify an attribute name that has not been defined in your model, you can select the **Selected in the list** option in the **External File Schedule** dialog. SIMPROCESS will add it to the model.

If a record contains any syntax error or undefined model element (without using **Selected in the list** option), the entire event record is considered invalid and is bypassed.

Examples

1. Generate `GreenDot` entities at 3 a.m. on January 1, 2014. Generate the number of entities defined in the **Generate Activity Properties** dialog.

```
01/01/2014 03:00:00 ENTITY:GreenDot
```

2. Generate six `RedDot` entities 7 hours after the simulation begins.

```
7.0 QUANTITY:6 ENTITY:RedDot
```

3. Generate four `RedDot` entities 7 hours after the previous entity generation.

```
+7:00 QUANTITY:4 ENTITY:RedDot
```

4. Generate a Truck entity at 2 a.m. on January 3, 2014. Assign a value of 50,000 to user attribute *Weight*, and the string "Yo, Heavy" to user attribute *Tag*:

```
01/03/2014 2:00 ENTITY:Truck ATTR:Weight=50000 Tag="Yo, Heavy"
```

5. Generate more Truck entities 4 hours after the previous event. Generate the number of entities defined in the `Generate Activity Properties` dialog:

```
+4.0 ENTITY:Truck
```

6. Generate two `RedDot` entities at 4 a.m. on January 3, 2014:

```
01/03/2014 04:00:00 ENTITY:RedDot QUANTITY:2
```

Notes on the Examples

If the examples are processed during a simulation which begins at midnight, January 1, 2014, the events occur at the following times:

1. January 1, 2014, 3:00 a.m.
2. January 1, 2014, 7:00 a.m.
3. January 1, 2014, 2:00 p.m. (14:00)
4. January 3, 2014, 2:00 a.m.
5. January 3, 2014, 6:00 a.m.
6. Never. The specified time is earlier than the time of the previous event, so the record is discarded.

APPENDIX H

Simulation Results File

The Simulation Results file is generated from the **File** menu. Select **File/Export/Simulation Results** to open the **Save Statistics** dialog. The Simulation Results file will, by default have a `.xpt` file extension and be saved to the current model directory. This file is tab-delimited, and can be opened using a text editor or spreadsheet.

The file will contain the complete statistical measures corresponding to the reports selected for the current model. This file give you raw statistical data that is displayed in the Standard and Custom Reports, in a standard format that can be opened by many different applications.

Format of the Simulation Results File

Each line of the exported simulation results file is a statistic recorded for objects such as Entity Types, Activities, or Resources. Each record contains the following fields or columns:

Owner Type

This is the type of the object that owns this statistic record. The four types are:

- **Entity**: representing an Entity Type
- **Activity**: representing an Activity
- **Resource**: representing a Resource
- **TimeStamp**: representing a Time Stamp
- **Attribute**: representing an Attribute

Owner Name

This is the full hierarchical name of the specific object that owns this statistic record.

Name

This is the name of the specific object that owns this statistic record.

Stat Type

This is the type of statistic displayed. Contained in the third column from the left, the Stat Type is an abbreviation for the name of the performance measure whose parameters follow to the right. See “Statistic Types” for a complete description of the types of statistics.

Replication

This number is the replication that the statistic was collected from. Any number, 1 or greater, refers to a specific replication.

If the replication number is 0, the statistics are sum of all replications. An -1 means that this statistic is the average of all replications.

Minimum

This is the smallest value that has been recorded for this statistic.

Maximum

This is the largest value that has been recorded for this statistic.

Sum

It is the total of all observed values for the statistic (i.e. X_i), if this is observation based.

For time weighted, it is the total of each observed value multiplied by the time that this value has persisted in (i.e. $X_i * \Delta T$).

Sum of Squares

It is the summation of the square of each observed values (i.e. $\sum(X_i^2)$), if this is observation based.

For time weighted, it is the summation of the square of each observed value multiplied by the time that this value has persisted in (i.e. $\sum(X_i*\Delta T)^2$).

Count

This is the number of observations recorded when the statistic was collected.

Average

This is the average of the statistic.

Std Deviation

This is the standard deviation of the statistic.

Statistic Types

The table below gives the descriptions of the statistic types within SIMPROCESS.

Labels	Type of Statistics	Descriptions
Entities:		
tokendelay	Observation based	Total cycle time (i.e. sum of wait, hold, and process)
tokenwaitdelay	Observation based	Time waiting for available resource
tokenholddelay	Observation based	Time waiting for condition to be reached
tokenprocessdelay	Observation based	Time processing (i.e. at specified duration)
tokentraveldelay	Observation based	Time traveling across connectors.
tokenlevel	Time-weighted	Number of entities in system
tokenwaitlevel	Time-weighted	Number of entities waiting for available resource
tokenholdlevel	Time-weighted	Number of entities waiting for condition to be reached
tokentravellevel	Time-weighted	Number of entities traveling across connectors.
tokenprocesslevel	Time-weighted	Number of entities in process (i.e. at duration)

Labels	Type of Statistics	Descriptions
tokentotalborn	Only the last value is recorded	Number of entities being generated
tokentotalalive	Only the last value is recorded	Number of entities still residing in the system
tokentotalkilled	Only the last value is recorded	Number of entities destroyed
UDV.delay	Observation based	Statistic of the specified attribute
UDV.level	Time-weighted	Statistic of the specified attribute
Resources:		
resrccap	Time-weighted	Units of resources (i.e. capacity)
resrcidle	Time-weighted	Units of idle resources
resrcbusy	Time-weighted	Units of busy resources
resrcmaintenance	Time-weighted	Units of resources not available (Unplanned Downtime)
resrcdown	Time-weighted	Units of resources not available (Planned Downtime)
resrcreserved	Time-weighted	Units of reserved resources
byactivitybusy.level	Time-weighted	Units of busy resources at the specified activity
byshiftbusy.level	Time-weighted	Units of busy resources during the specified shift
Connectors:		

Labels	Type of Statistics	Descriptions
tokentotalin	Only the last value is recorded	Number of entities entered this connector
tokentotalhere	Only the last value is recorded	Number of entities still residing in this connector
tokentotalout	Only the last value is recorded	Number of entities that have left this connector
bytokenin	Only the last value is recorded	Number of entities of the specified type that entered the connector
bytokenhere	Only the last value is recorded	Number of entities of the specified type still residing in the connector
bytokenout	Only the last value is recorded	Number of entities of the specified type that have left the connector
tokendelay	Observation based	Total cycle time at this connector
bytoken.delay	Observation based	Total cycle time for the specified entity type at this connector
tokenlevel	Time-weighted	Number of entities in this connector
bytoken.level	Time-weighted	Number of entities of the specified type in this connector
Activities:		
tokentotalin	Only the last value is recorded	Number of entities entered this activity

Labels	Type of Statistics	Descriptions
tokentotalhere	Only the last value is recorded	Number of entities still residing in this activity
tokentotalout	Only the last value is recorded	Number of entities that have left this activity
tokentotalearlyout	Only the last value is recorded	Number of entities that left the activity without processing
bytokenin	Only the last value is recorded	Number of entities of the specified type that entered the activity
bytokenhere	Only the last value is recorded	Number of entities of the specified type still residing in the activity
bytokenout	Only the last value is recorded	Number of entities of the specified type that have left the activity
bytokenoutearly	Only the last value is recorded	Number of entities of the specified type that left the activity without processing
tokendelay	Observation based	Total cycle time (i.e. sum of wait, hold, and process) at this activity
tokenwaitdelay	Observation based	Time waiting for available resource at this activity
tokenholddelay	Observation based	Time waiting for condition to be reached at this activity

Labels	Type of Statistics	Descriptions
tokenprocessdelay	Observation based	Time processing entities (i.e. in the specified durations) at this activity
bytoken.delay	Observation based	Total cycle time (i.e. sum of wait, hold, and process) for the specified entity type at this activity
bytokenwait.delay	Observation based	Time waiting for available resource for the specified entity type at this activity
bytokenhold.delay	Observation based	Time waiting for condition to be reached for the specified entity type at this activity
bytokenprocess.delay	Observation based	Time processing entities (i.e. in the specified durations) for the specified entity type at this activity
tokenlevel	Time-weighted	Number of entities at this activity
tokenwaitlevel	Time-weighted	Number of entities waiting for available resource at this activity
tokenholdlevel	Time-weighted	Number of entities waiting for condition to be reached at this activity
tokenprocesslevel	Time-weighted	Number of entities in process (i.e.at duration) at this activity
bytoken.level	Time-weighted	Number of entities of the specified type at this activity

Labels	Type of Statistics	Descriptions
`bytokenwait.level`	Time-weighted	Number of entities of the specified type waiting for available resource at this activity
`bytokenhold.level`	Time-weighted	Number of entities of specified type waiting for condition to be reached at this activity
`bytokenprocess.level`	Time-weighted	Number of entities of the specified type in process (i.e.at duration) at this activity
`UDV.delay`	Observation based	Statistic of the specified attribute
`UDV.level`	Time-weighted	Statistic of the specified attribute
Time Stamps:		
`tokendelay`	Observation based	Time elapsed between Time Stamps

APPENDIX I

UML Interfaces

SIMPROCESS and Unified Modeling Language (UML) tools complement each other by bridging the gap from business analysis to systems analysis. SIMPROCESS is used for the business process models, and UML tools are used for the systems models. These tools work together in documenting the operational architecture of a business. They communicate the content of the business requirements and the systems requirements and designs by using both notations in combination.

UML-based tools such as Rational Rose focus on Systems Modeling to help define requirements, model designs, and develop object-oriented code such as Java. Rational Rose is one of the leading tools on the market for system modeling activities in a software development life cycle, but other capable tools support the UML standard.

UML tools do not fully address the functional/business process modeling requirements. For example, Use Cases and Activity Diagrams are the only views UML has for modeling any part of the business process, but they still focus on the "system" needs in the models rather than the function or business steps. For instance, Use Cases show Actors (usually people) interacting with objects (usually realized as object oriented components such as Java classes). This helps to understand the system transactions but does not model the full breadth of the business workflow or process. Activity diagrams similarly model activity steps that can be used to model business flow diagrams but typically focus on objects and transition of states between objects based on changes to the objects in a system transaction. Activity diagrams do not provide the dimensions needed to do robust business process analysis such as dynamic modeling, entity flow simulations, resource utilization, queueing theory, and cost based business metrics.

SIMPROCESS focuses on the functional and business work flow (including manual/physical processes that have nothing to do with the system but need to be understood and modeled in order to build a system to support the business properly) and not just on the system transactions. This is made clear when looking at how activities such as delays, splits, joins, batching, unbatching, assembling, etc., are described for behavior in a SIMPROCESS model. These are representative of business activities that people do in carrying out their jobs regardless of what interaction they have with the system. In addition, SIMPROCESS looks at the resources, entities, and workflow as a complete dynamic model that allows visualization of how the business works both from manual steps and system interaction. This is important in order to match up the business transactions with the system transactions.

SIMPROCESS has a more functional and business view and focuses on activity based costing metrics, throughput, bottlenecks, timing, re-work and other business performance metrics regardless of the system that is supporting the business. This is why business and functional representatives prefer SIMPROCESS for the business modeling - it speaks their language. SIMPROCESS focuses on the business concerns and not all the system modeling notations and methods (which often confuses functional people). The dynamic modeling capabilities for business metrics are non-existent in UML modeling tools. This is appropriate since UML tools are system analysis, design, and development tools, not process and workflow modeling tools.

There is a clear gap between business analysis and systems analysis. SIMPROCESS focuses on the business analysis and UML tools focus on the systems analysis. Clearly, both are needed from business requirements to systems requirements to system design to system development. They work hand in hand and not in opposition to each other. This gap in business analysis and systems analysis is the basis for the SIMPROCESS to UML interfaces. These interfaces use the complementary strengths of both UML and SIMPROCESS to provide customers a complete toolset for business and systems modeling.

SIMPROCESS has two basic interfaces to UML. One is the ability to export a SIMPROCESS model to a UML Activity Diagram. The export to Activity Diagrams is based on the standard UML XMI specification and provides a transfer of the SIMPROCESS activities and connectors to UML activities and transitions. In addition, SIMPROCESS Entities and Resources are exported as UML Classes (with attributes included); however, the classes are not accompanied by a UML Class Diagram. The Activity Diagram includes the Diagram information for Rational Rose only. The XMI activity information can be imported into any UML tool other than Rational Rose that supports XMI by simply deleting the Diagram information at the bottom of the XMI file that is created from the export process.

The other type of interface is an active link between activity nodes on a SIMPROCESS model and Use Case Diagrams in a Rational Rose model. This interface requires a Microsoft Windows version of SIMPROCESS and a licensed copy of Rational Rose.

Each type of interface is described in more detail in the following sections.

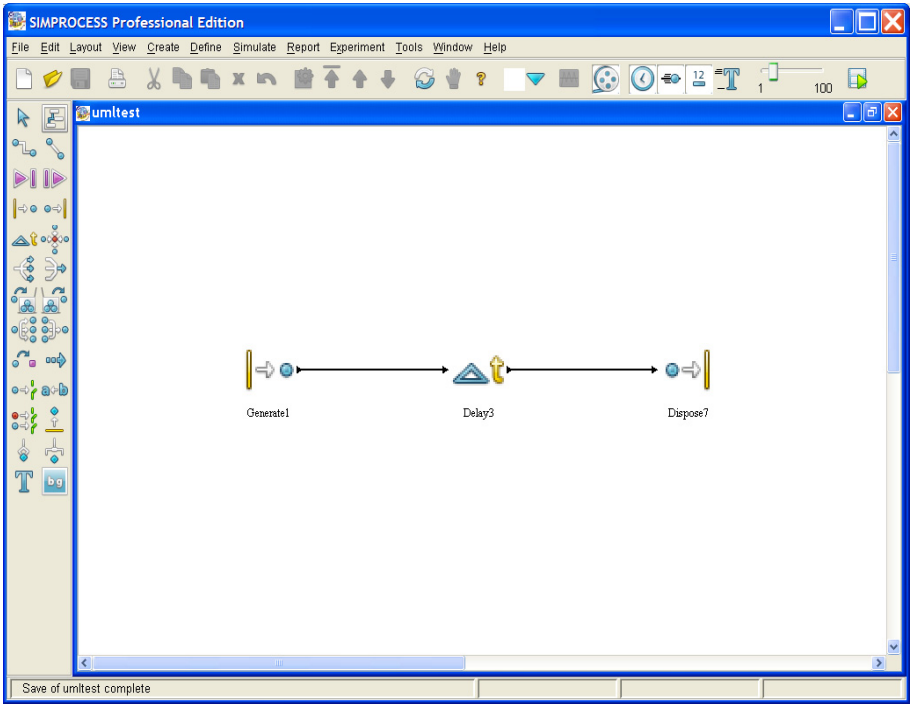
Exporting to UML

This capability can be used by selecting **File/Export/UML Activity Model** from the SIMPROCESS menus. It outputs the model to a UML compatible XML file. No dialog is displayed for this export operation. The file is created in the model's directory with a `.xml` extension and is compatible with Rational Rose, including diagram information. The XMI file may be compatible with other UML tools as well, such as the ArgoUML open source tool, but diagram information intended for Rational Rose will not be compatible with other UML tools.

To import the model into Rational Rose, Rational Rose must have the Import/Export XMI plug-in installed. This plug-in can be obtained from the Rational Rose web site (www.rational.com). Once it is installed, simply choose the **Import UML Model** from the **Tools** menu in Rational Rose.

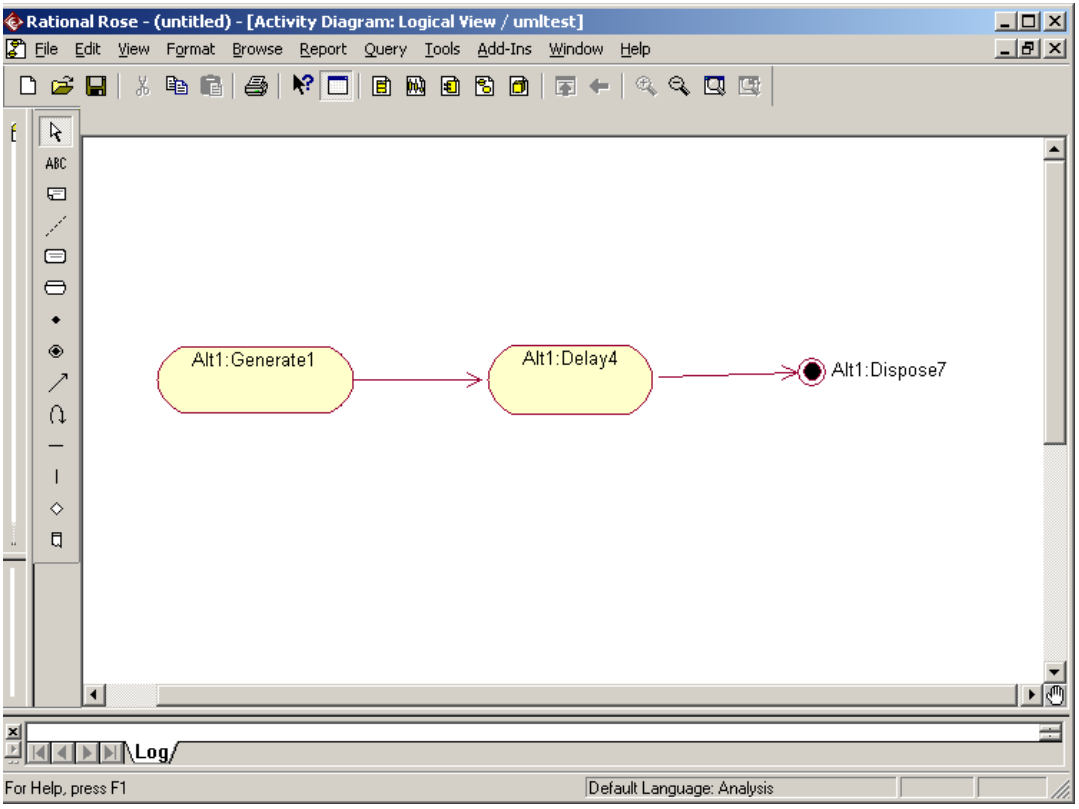
In some cases, due to slight differences in the aspect ratio of the graphics in SIMPROCESS and the Rational Rose tools, minor visual clean up of the imported Activity Diagram may be required.

The following is a simple SIMPROCESS model that is exported to UML:



The above SIMPROCESS model would look like the following in Rational Rose:

Appendix I - Exporting to UML

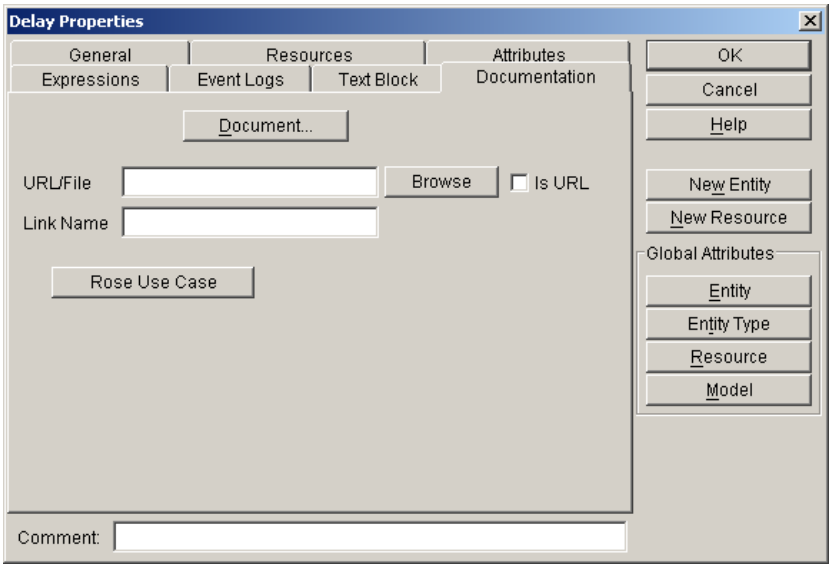


Rose Use Cases

SIMPROCESS provides the capability for the user to connect nodes (activities) on the SIMPROCESS model directly to a Rose Use Case Diagram. This allows for a seamless flow from business process modeling and simulation analysis in SIMPROCESS to systems models in Rose. Exported HTML views of the SIMPROCESS models (Export) are directly linked to Rational Rose exported HTML models. This gives stakeholders a seamless wide area review of the business process models with Rose Use Case models. SIMPROCESS models typically include both manual process steps (activities) and steps that are automated with information technology. This capability ties the manual process steps and the automated process steps (supported by Use Case links) together so that the system requirements are in complete context with the business process.

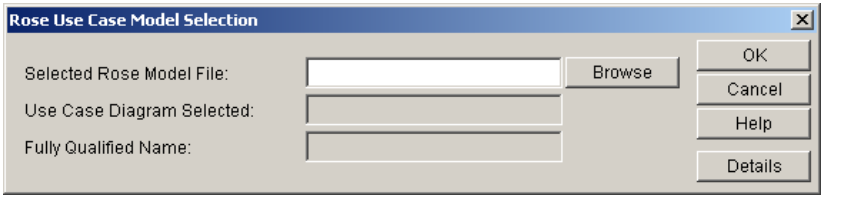
This feature is licensed by CACI separately from SIMPROCESS and is only available with the Windows version of SIMPROCESS. For this feature to be active, the additional license must be purchased, and Rational Rose must be installed.

Rose Use Case links are added by selecting the **Rose Use Case** button on the **Documentation** tab on any activity properties dialog. This button will only be enabled if the above criteria are met.



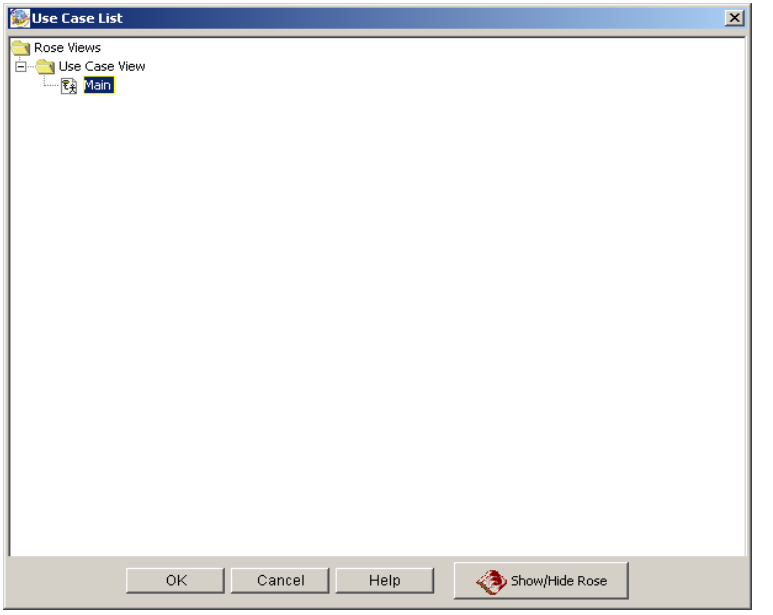
Clicking **Rose Use Case** brings up the following dialog. The path to a Rose model file can be entered or the **Browse** button can be used to select a Rose model file. Once a Rose model has been selected, a reference to it is stored in the SIMPROCESS model file for subsequent use (this reference will be invalidated if the file is later moved or renamed). The connection to a Rose model remains until the

model file name is cleared, and the **OK** button is clicked.



Once a Rose model is connected to the SIMPROCESS model, the **Details** button will connect SIMPROCESS to a licensed copy of Rational Rose using the Rational Rose Automation capability. If a successful connection and load of the Rose model is completed (this occurs in the background), a window like the one shown below is presented listing all the Rose views and subordinate packages that contain Use Case Diagrams. Note that if a package or view does not contain any Use Case Diagrams, it is not included in the window since the SIMPROCESS connection is to Use Case Diagrams only.

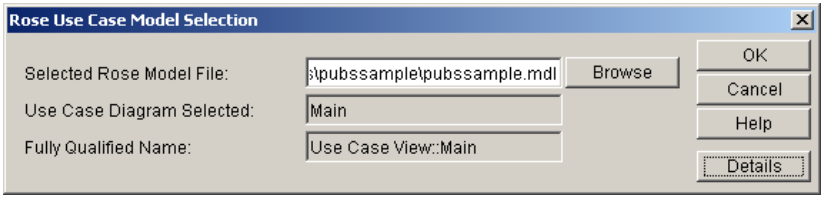
Utilizing this window, a Use Case Diagram can be navigated to and selected to associate with the SIMPROCESS activity node. Note that the structure in the dialog is a simple Tree structure that works like other graphical file directory trees.



A single Use Case Diagram can be selected by clicking on it and then clicking the **OK** button. This

action will assign that Use Case Diagram to the SIMPROCESS activity node. Optionally, the **Show/Hide Rose** button can be selected to bring the Rose tool and model up on the screen so that it is available. Once Rose is visible the Rose model can be modified as desired, and the Rose model can be saved. However, any new Use Case Diagrams added will not automatically appear in the SIMPROCESS Use Case List window until it is closed and reopened.

After a Use Case Diagram is selected and the **OK** button is clicked, the Use Case Diagram information will be added into the SIMPROCESS dialog, as shown below. The Use Case Diagram name and fully qualified name (which includes the entire package structure in Rose) are not directly editable. To clear the Use Case Diagram connection, simply select and clear the **Selected Rose Model File** field in this dialog and click **OK**. To change the **Use Case Diagram Selected** (the node to which SIMPROCESS is connected), simply click on the **Details** button and choose another Rose Use Case Diagram from the Use Case Diagram dialog as described above.



An important feature of this Use Case Diagram connection between SIMPROCESS and Rational Rose is that when the Rose model is published to HTML and the SIMPROCESS model is published to HTML, the SIMPROCESS HTML nodes will be automatically linked to the Rose Use Case Diagrams in the HTML. For this feature to work, the Rose published model must be in a directory named `rose` within the directory of the SIMPROCESS published model on the disk drive or web server. When navigating the SIMPROCESS published HTML model, moving the mouse pointer over a SIMPROCESS activity node that has been connected to a Rose Use Case Diagram as described in this section gets a Javascript popup that links directly to the Rose published HTML Use Case Diagram automatically. This capability is very useful for wide area review of business process models that have Use Case functional requirements attached to certain nodes in the SIMPROCESS model. Hence, the business process requirements and the system requirements are in context of each other using this technique.

APPENDIX J

Running Models Without GUI

SIMPROCESS models can be run without the SIMPROCESS Graphical User Interface (GUI). There are two executables in the SIMPROCESS directory: SIMPROCESS and SPRunSimulation (any extensions are platform specific). The SIMPROCESS program starts the SIMPROCESS GUI, required to build and edit models. This GUI is typically used to run models as well. However, SPRunSimulation can be used to run a SIMPROCESS model without the GUI. A valid `license.dat` file is required to run SIMPROCESS and SPRunSimulation. The same license file is used for both.

Since access to the GUI is not available, results of the run must be automatically output to the SIMPROCESS database (see "SIMPROCESS Database"), user created files, or a user created database. Statistics collected during the run can be accessed using System Methods. (See "Accessing Statistics During Simulation".) The Standard Report can be generated automatically after the run by selecting **Generate Standard Report after run** on the **Statistics Collection** dialog. (See "Generating Standard Reports From Experiments".)

To run a model with SPRunSimulation, either execute commands from a command prompt or create a batch file or a shell script. The command must include SPRunSimulation and the complete path to the model that is to be run, as shown in this example for Windows.

```
SPRunSimulation "C:\Program
Files\SIMPROCESS\models\Demos\Purchasing.spm"
```

The above is the simplest form for running a model without the GUI. However, model parameters (see "Globally Defining an Attribute from the Menu") and process alternatives can be set as well as the number of replications. In addition, the simulation results can be automatically placed in the

SIMPROCESS database, and the model can be saved. These options can be set either on the command line or by using a properties file.

Available Options

The following options can be set on the command line or in a properties file. The format of the option is always the name of the option followed with no spaces by an equals sign followed with no spaces by the value.

- Any global attribute designated as a model parameter - Model.NumResource1=2
- Any process alternative - "Process name=Alternative name". If the process is a Master Process (Master Processes), the specified alternative for the Master Process will be applied to all copies.
- Number of replications - Replications=5
- Design for committing the results to the SIMPROCESS database - "Design=My Design"
- Scenario for committing the results to the SIMPROCESS database (must be used if Design is used) - "Scenario=My Scenario"
- Option to save the model after the parameters are set (thus, the next time the model is opened in SIMPROCESS, the model parameters set by command line or properties file retain their latest value) - Save=True (the absence of this option is equivalent to Save=False)
- Can set the starting and/or ending date and time of the model. Absence of the time portion indicates midnight. - Start=1/1/2005, End=1/1/2006 08:00:00.

Note: Design and Scenario should not be used on Linux since the SIMPROCESS database is an Access database and is not available for Linux systems. Also, Design and Scenario should not be used on Windows if MS Access is not installed.

Setting Options on Command Line

Any parameters to be set must follow the model on the command line. For example

```
SPRunSimulation "C:\Program
Files\SIMPROCESS\models\Demos\Purchasing.spm"
Model.NumProdAPersonnel=6 Model.NumProdBPersonnel=4
"Purchasing=Centralized, Functional" Replications=2 "Design=2 Reps"
"Scenario=Scenario 1" Save=True
```

Parameters that have spaces must be in quotes. The order of the parameters following the model is not important. If Design and Scenario are omitted, then the results are not committed to the database.

Setting Options Using Properties File

The preferred method for setting parameters is to use a properties file. The file can have any name as long as the extension is `.properties`. When using a properties file, the properties file name follows the model on the command line.

```
SPRunSimulation "C:\Program
Files\SIMPROCESS\models\Demos\Purchasing.spm" "C:\Program
Files\SIMPROCESS\models\Demos\Purchasing\Purchasing.properties"
```

This example shows a file named `Purchasing.properties` that is in the directory of the **Purchasing.spm** model. The following is an example of a properties file. Three model parameters are set, one process alternative is set (Purchasing), the number of replications is set, and the Design and Scenario are set. The order is not important. Do not include a Design and Scenario if the results of the run should not be committed to the SIMPROCESS database.

```
Model.NumProdAPersonnel=6
Model.NumProdBPersonnel=4
Model.NumProdCPersonnel=4
Purchasing=Centralized, Functional
Scenario=s1
Design=New Design
Replications=2
```

APPENDIX K

SIMPROCESS and External Java Classes

SIMPROCESS is a Java application. During execution, Java programs use the "classpath" to locate individual classes. In order to use some of the SIMPROCESS special features that use external Java classes, certain aspects of the classpath need to be understood.

There are three specific ways that the classpath can be used to support external Java classes in SIMPROCESS.

- Java Extension Mechanism
- Model-Specific Java Additions
- ext Directory

Please note that there are specific recommendations for when to use one over another, as not all of them are equal.

Java Extension Mechanism

The Java Extension Mechanism provides a way of adding things to the classpath without changes to any files installed with SIMPROCESS. To use it, JAR files containing needed Java packages must be placed where the Java Runtime Environment (JRE) will seek them when it needs to resolve a reference. Since SIMPROCESS installs a JRE for its own use, that location will always be the `jre/lib/ext` directory created by the installer.

Typical Use

The most common use of the Java Extension Mechanism is to support using databases other than Microsoft Access with SIMPROCESS. (See "SIMPROCESS and Other Databases".) In the example cited there, database drivers for MySQL, a popular open source database, are required in order for SIMPROCESS to use it, so one or more JAR files must be placed in this location.

Recommended Use

When third party Java packages are needed by SIMPROCESS, the JAR files containing them can be placed in this location. However, it's recommended that this be done only when specifically necessary and when the model-specific mechanism described in the next section is not appropriate. In particular, database drivers must use this mechanism, as they are handled by SIMPROCESS itself. Therefore, the database drivers will not be sought in model directories.

Warning on Potential File Loss

When adding JAR files to the `jre/lib/ext` directory of an installed copy of SIMPROCESS, it's a good idea to keep copies of the original JAR files. When SIMPROCESS is uninstalled, the uninstaller program will unconditionally remove the entire `jre` directory, thus removing any added JAR files. In addition, every new release installer will install a `jre` directory. It's possible that this can result in the loss of JAR files added here. So any such additions should be double-checked any time a new installer is used.

Model-Specific Java Additions

As of release 4.4, SIMPROCESS provides a new way to use added Java classes. Except in special circumstances (e.g., database drivers), this is the preferred way of adding Java code to SIMPROCESS.

When a model is opened by SIMPROCESS its model directory is created, if not already present. To use user-provided Java classes with the model, a new subdirectory named `classes` must be created in the model directory. When simulating the model, the contents of this subdirectory will be included in the classpath. The `classes` directory can contain:

- Any number of JAR files containing packaged Java classes and related resources to which the model may refer
- Compiled Java classes in directories representing their package structure, along with any related resources to which the classes may refer
- Individual compiled Java class files considered to be in the "default package" (although this is not recommended)

JAR files **and** class files can be used simultaneously with a model.

When to Use classes Subdirectory

The **`ExternalCall`** System Method supports the use of Java classes in this subdirectory (see "Method ExternalCall") . Others may be added in the future. As recommended above, the `classes` subdirectory should be used in preference to either of the other described mechanisms (Java Extension Mechanism and ext Directory) except when specifically necessary or impractical not to do so.

Why Use classes Subdirectory

Even if multiple models need to share the same set of classes and/or JAR files, this subdirectory should be used in order to make model sharing easier. Since a model can carry its own background and icon images (see "Importing Graphics Image Files") and carry its own external Java classes, the need for instructions on how to place class files and import images is eliminated. Instead, the model file and its like-named directory can be packaged and sent as a single unit. The recipient can then open the model and simulate without having to import background and icon images or place external Java classes in specific locations.

ext Directory

The `ext` directory (see "ext Directory") is included in the SIMPROCESS classpath. This means Java code placed here will be accessible to SIMPROCESS when appropriate. The code should be compiled and its package directory structure copied into the `ext` directory. It is possible to use code without package statements, in what Java refers to as the "default" package, but this is not recommended and not "officially supported" by Sun, though it will work.

In SIMPROCESS releases through 4.3, this directory was the only means of using "loose" Java classes (those not in a JAR file) with SIMPROCESS. Java classes placed here were typically used with the **`ExternalCall`** System Method and other similar features. This directory is still supported for such use, but it is recommended that it only be used when multiple models need to share the same classes (not necessarily simultaneously) and when it's impractical to use the newer model-specific mechanism described in the previous section. Note that only "loose" Java classes may be used in this directory. To use Java classes contained a JAR file, use one of the other means described above.

D
Distribution
Beta 43
Erlang 44
Exponential 41
Gamma 42
Geometric 50
Hyperbolic 51
Lognormal 46
Normal 38
Poisson 47
Triangular 39
Uniform 37
Weibull 45
S
Statistical Distributions 35